Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Applied Time Series Analysis

FS 2012

Dr. Marcel Dettling

Institute for Data Analysis and Process Design

Zurich University of Applied Sciences

CH-8401 Winterthur

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, ...

The purpose of time series analysis, simply put, is to understand the past data, and to forecast 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 gives rise to the observed 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, both of which can be modeled deterministically by mathematical functions of time. 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
```

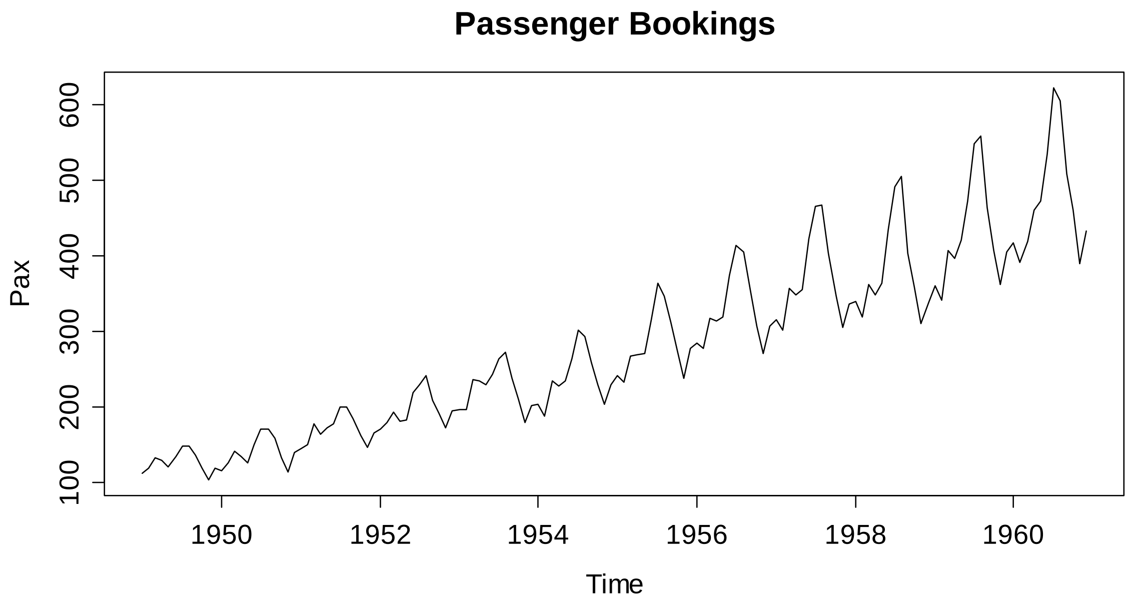
Some further information about this dataset can be obtained by typing `?AirPassengers` in **R**. The data are stored in an **R**-*object* of *class* `ts`, which is the specific class for time series data. However, for further details on how time series are handled in **R**, we refer to section 3.

One of the most important steps in time series analysis is to visualize the data, i.e. create a time 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 4.2.

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.



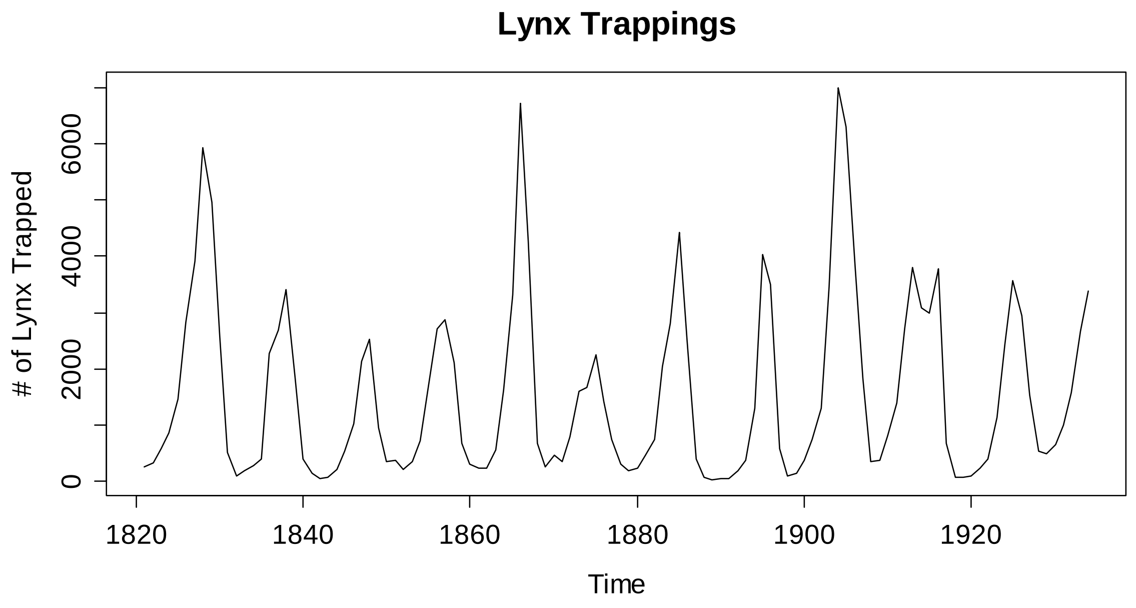
We can naturally attribute the increasing trend of the series to causes such as rising prosperity, greater availability of aircraft, cheaper flights and increasing population. The seasonal variation coincides strongly with vacation periods. For this reason, we here consider both trend and seasonal variation as deterministic components. As mentioned before, section 4.2 discusses visualization and estimation of these components, while in section 6, 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 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. There is no fixed natural period which suggests these results. Thus, we will not attribute this behavior not to a deterministic periodicity, but to a random, stochastic one.



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

1.2.3 Luteinizing Hormone Measurements

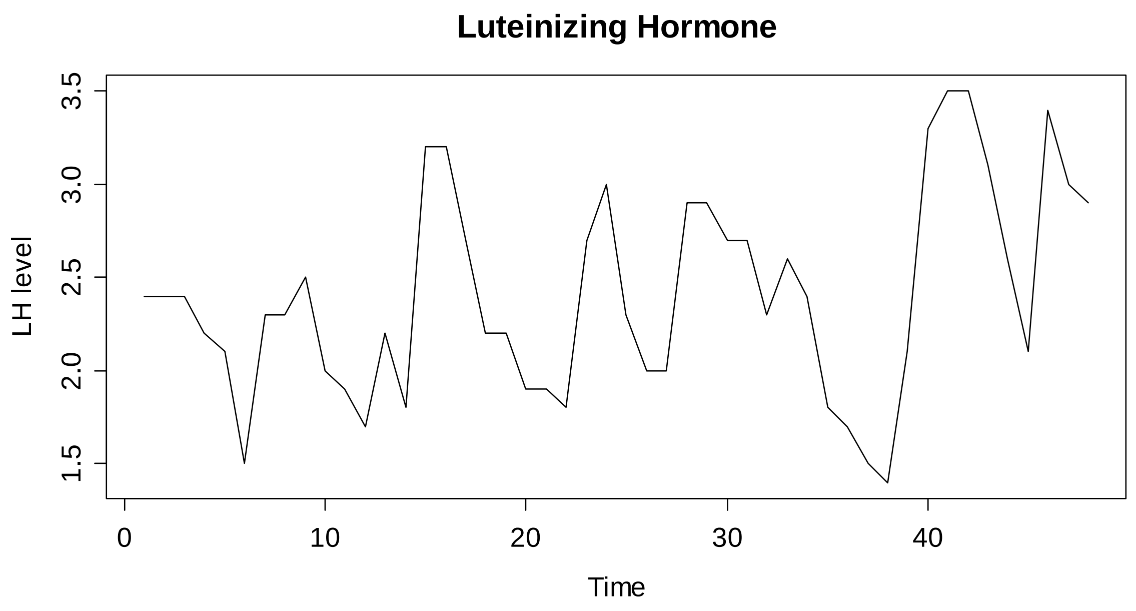
One of the key features of the above lynx trappings series is that the observations apparently do not stem from independent random variables, but there is some serial correlation. If the previous value was high (or low, respectively), the next one is likely to be similar to the previous one. To explore, 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
```

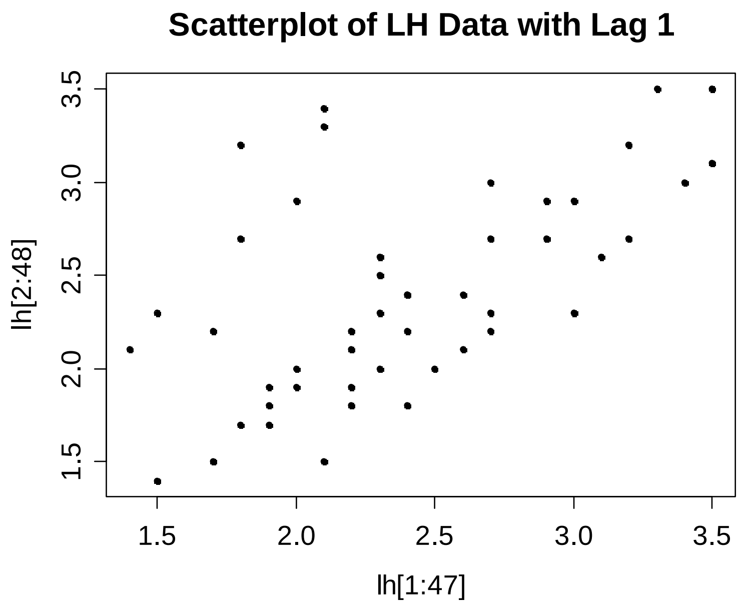
Again, the data themselves are of course needed to perform analyses, but provide little overview. We can improve this by generating a time series plot:

```
> 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 variation. 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 again, 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
```

This figure is an estimate for the so-called autocorrelation coefficient at lag 1. As we will see in section 4.3, 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

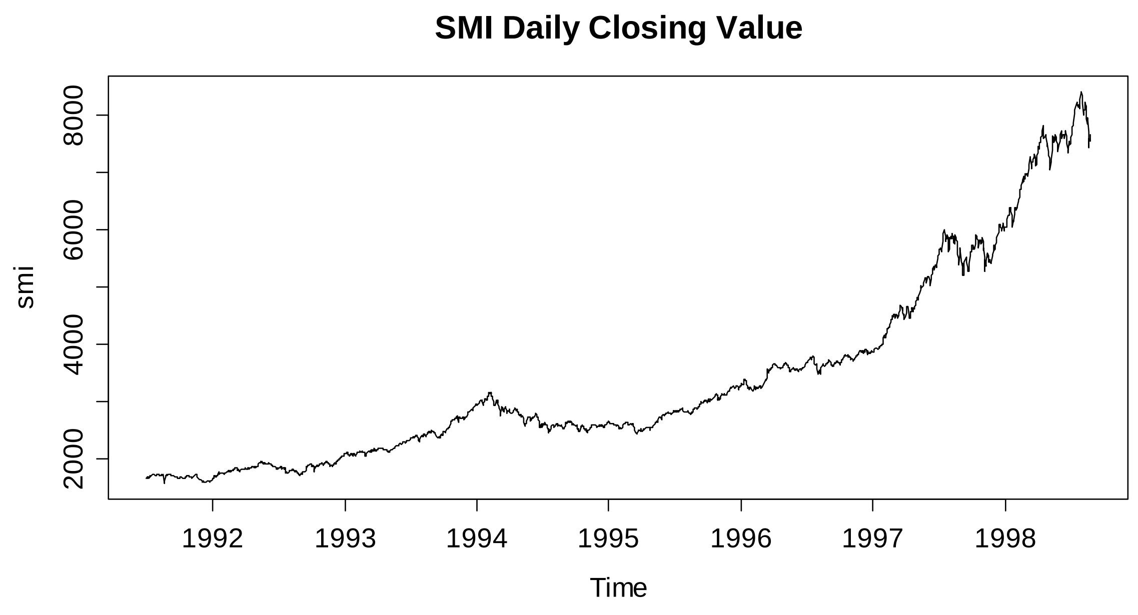
The SMI is the blue chip index of the Swiss stock market. It summarizes the value of the shares of the 20 most important companies, and contains around 85% of the total capitalization. Daily closing data for 1860 consecutive days from 1991-1998 are available in **R**:

```
> 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:

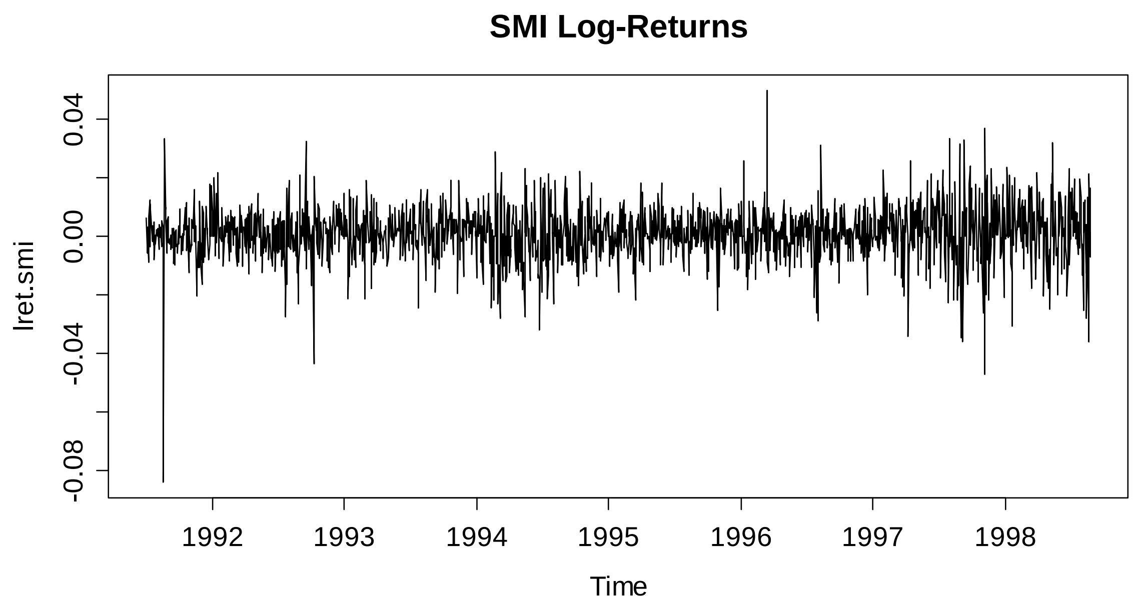
```
esm <- EuStockMarkets
tmp <- EuStockMarkets[,2]
smi <- ts(tmp, start=start(esm), freq=frequency(esm))
plot(smi, main="SMI Daily Closing Value")
```

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 logged-value of today's value divided by the one of yesterday:

```
> tmp      <- log(smi[2:1860]/smi[1:1859])
> lret.smi <- ts(tmp, start=c(1991,131), freq=frequency(esm))
> plot(lret.smi, main="SMI Log-Returns")
```



The SMI log-returns are an approximation to the percent change with respect to the previous day. As can be seen below, they do not show a trend anymore, but show some of the stylized facts that most log-returns of financial time series have. Using lagged scatterplots or the correlogram (to be discussed later in section 4.3), 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 with respect to the 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 usual regression framework. However, the all-to-common assumption of (serially) uncorrelated errors is usually violated in a time series framework. 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.

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. As we all know, we cannot do statistics with a single observation. As a way out of this situation, we need to impose some conditions on the joint distribution function F.

2.2 Stationarity

The aforementioned condition on the joint distribution F is the concept of *stationarity*. In colloquial language this 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 s, t and k, the observations $x_t, \ldots, x_{t+k}$ could have just as easily occurred at times $s, \ldots, s+k$.

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 moments:

Expectation	μ_t	$= E[X_t]$,	for stationary series: $\mu_t = \mu$.
Variance	σ_t^2	$= Var(X_t)$,	for stationary series: $\sigma_t^2 = \sigma^2$.
Covariance	$\gamma(t_1, t_2)$	$= Cov(X_{t_1}, X_{t_2})$,	for stationary series: $Cov(X_t, X_{t+h}) = \gamma(h)$.

In other words, strictly stationary series have *constant expectation*, *constant 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.

In practice, 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 a single realization, i.e. an observed time series. We can, however, always check whether a time series process shows *constant 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's 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 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 pastime 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 4.3) 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 still 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
88	76	112	109	91	98	139

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 will not be a seasonal effect, which can only appear with 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)
> ts.dat <- ts(rawdat, start=2004, freq=1)
> ts.dat
Time Series:
Start = 2004
End = 2010
Frequency = 1
[1]  88  76 112 109  91  98 139
```

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] 2010    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 = 2010
Frequency = 1
[1] 2004 2005 2006 2007 2008 2009 2010
```

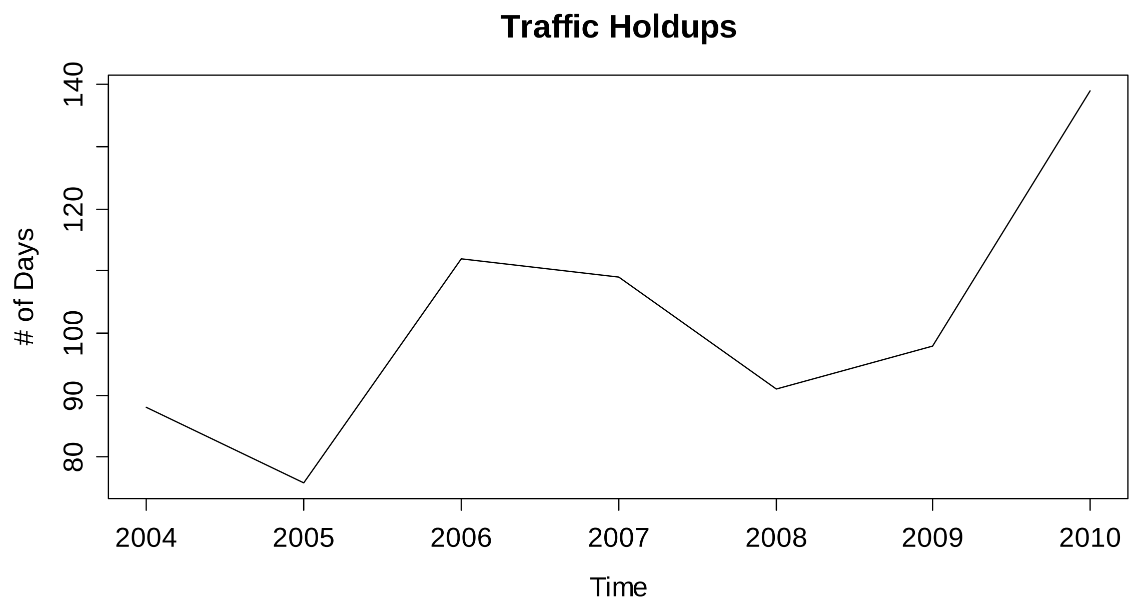
The next basic, but for practical purposes very useful function is `window()`. It is aimed at selecting a subset from a time series. Of course, also regular **R**-subsetting such as `ts.dat[2:5]` does work with the time series class. However, this results in a vector rather than a time series object,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

Besides the basic `ts` class, there are several more which offer a variety of additional options, but 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 dealing 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 are 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, as which they serve for purposes as visualization, for decomposition, or for 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 “a pile of numbers”, i.e. data, it is important to first gain an overview. In the context of time series analysis, this can be done in several ways. We start by discussing time series plots, then 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.

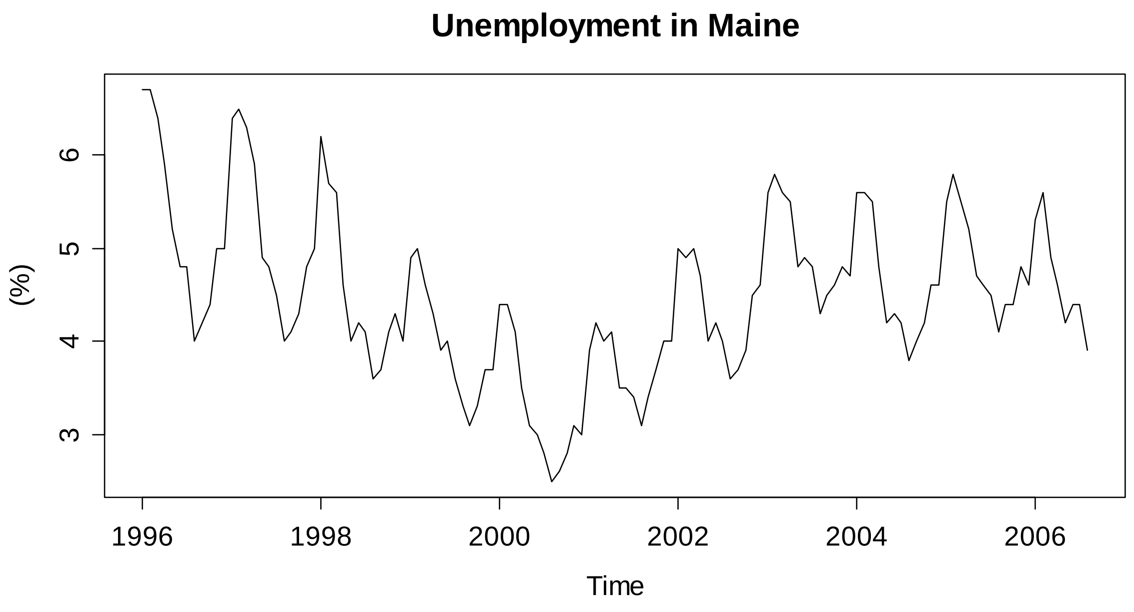
4.1 Visualization

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. We have seen several examples in section 1.2, where we also got acquainted with **R**’s generic `plot()` function that produces such output. We here show another example, the monthly unemployment rate for the US state of Maine, from January 1996 until August 2006. The data are available from a text file on the web. We can read it directly into R, define the data as an object of class `ts` and then do the time series plot:

```
> www <- "http://www.massey.ac.nz/~pscowper/ts/Maine.dat"
> dat <- read.table(www, header=TRUE)
> tsd <- ts(dat, start=c(1996,1), freq=12)
> plot(tsd, ylab="(%)", main="Unemployment in Maine")
```

Not surprisingly, the series shows both seasonal variation and a non-linear trend. Since unemployment rates are one of the main economic indicators used by politicians/decision makers, this series poses a worthwhile forecasting problem.



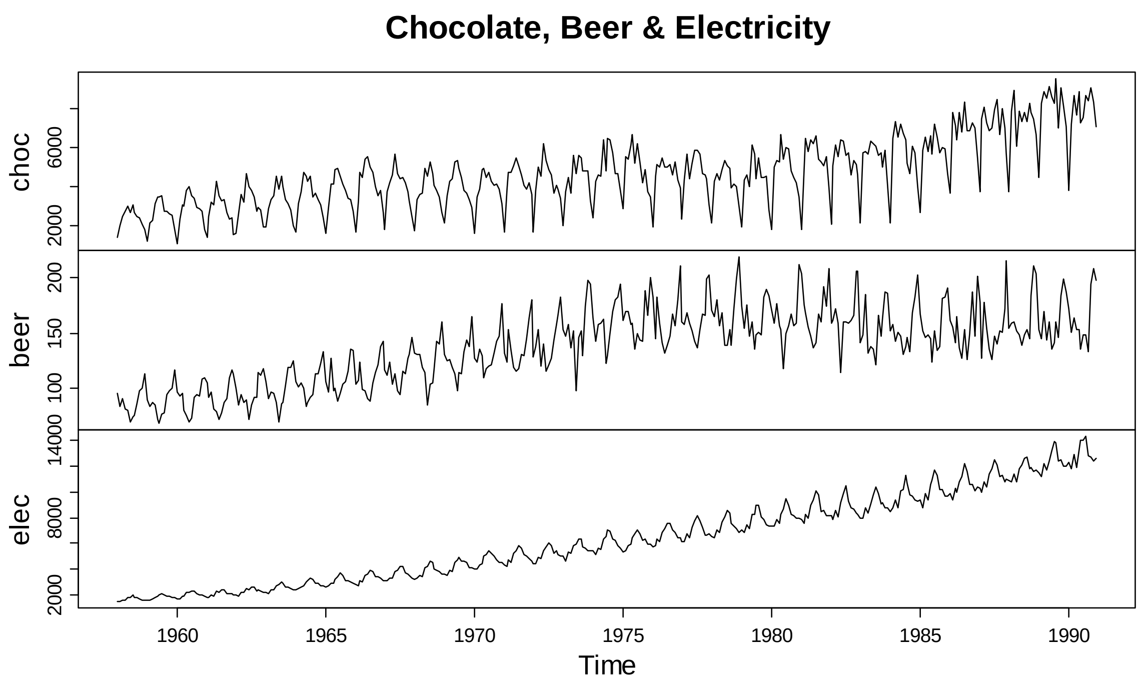
Another issue is the correct aspect ratio for time series plots: if the time axis gets too much compressed, it can become difficult to recognize the features of a series. Thus, we here recommend choosing the aspect ratio appropriately. Unfortunately, there are no hard and simple rules on how to do this. As a rule of the thumb, the “banking to 45 degrees” rule exists. This means that 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.

4.1.2 Multiple Time Series Plots

It is quite often the case that we encounter an applied problem where we are provided with multiple time series. Here, we illustrate some basics on how to import, define and plot them properly. Our example contains 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 are available from the Bureau of Australian Statistics and are, in pre-processed form, accessible as a text-file online.

```
www <- "http://www.massey.ac.nz/~pscowper/ts/cbe.dat"
dat <- read.table(www, header=TRUE)
tsd <- ts(dat, start=1958, freq=12)
plot(tsd, main="Chocolate, Beer & Electricity")
```

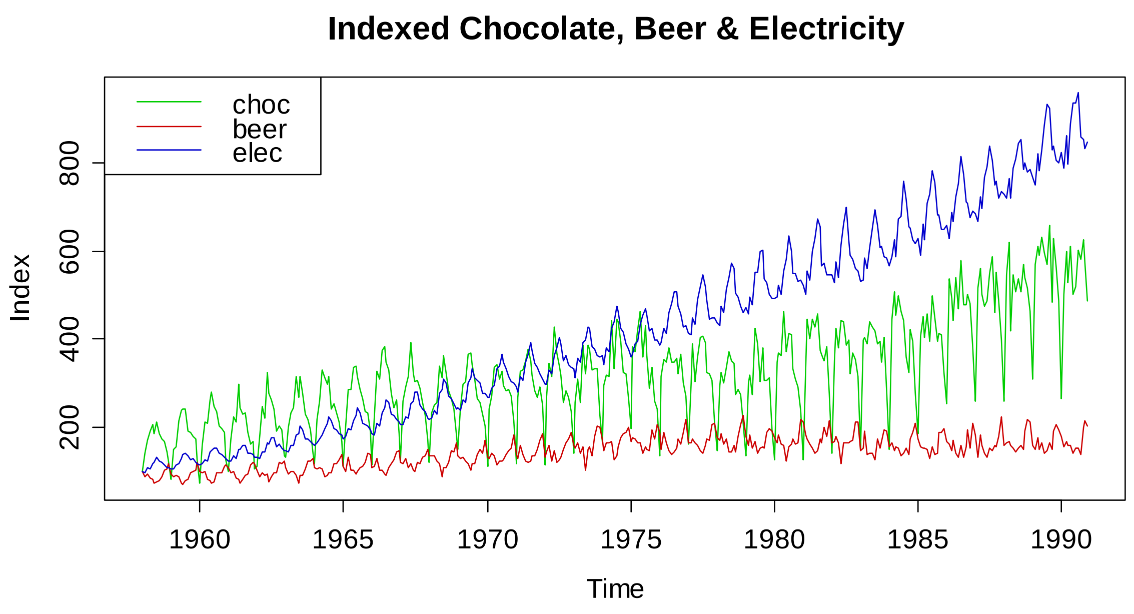
All three series show a distinct seasonal pattern, along with a trend. It also instructive to know that the Australian population increased by a factor of 1.8 during the period where these three series were observed.



As visible in the bit of code above, plotting multiple series into different panels is straightforward. As a general rule, using different frames for multiple series is the most recommended means of visualization. However, there may be other cases where it is 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. While R offers function `ts.plot()` to include multiple series in the same frame, that function does not allow color coding. For this reason, we prefer doing some manual work.

```
## Indexing the series
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.ts(tsd[,1], ylim=range(tsd), ylab="Index", col=clr[1])
title("Indexed Chocolate, Beer & Electricity")
lines(tsd[,2], col=clr[2])
lines(tsd[,3], col=clr[3])

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

In the indexed single frame plot below, 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.



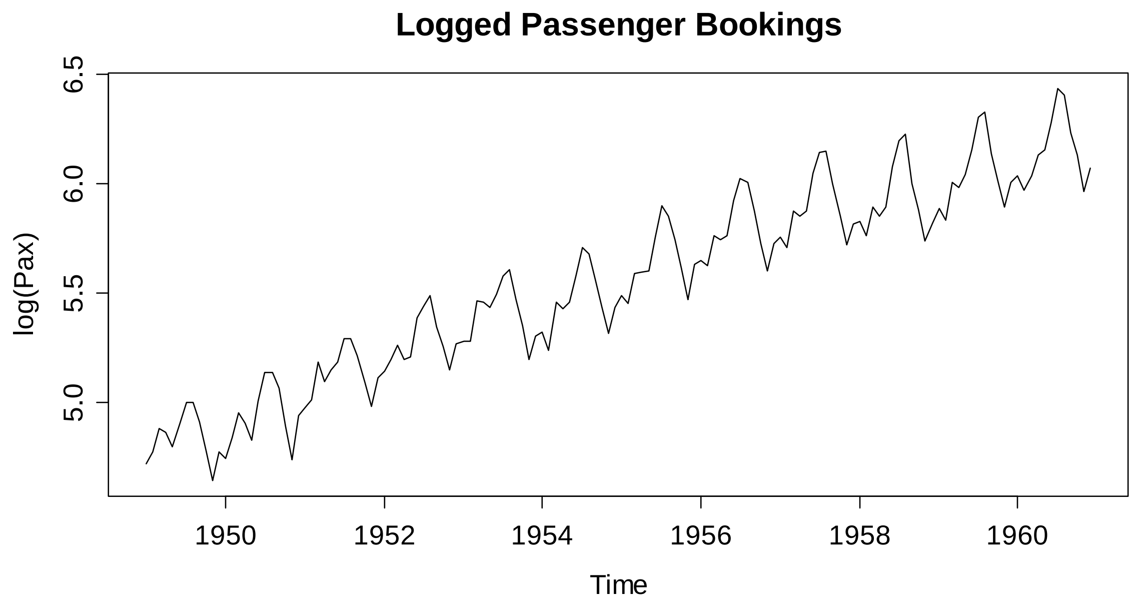
4.2 Decomposition

4.2.1 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 we treated so far have either shown a trend or a seasonal effect, and thus are non-stationary. In this section, we will learn how to deal with deterministic trend and seasonal variation. This is achieved by using *decomposition models*, the easiest of which is the *simple additive* one:

$$X_t = m_t + s_t + E_t ,$$

where X_t is the time series process at time t, m_t is the trend, s_t is the seasonal effect, and E_t is the remainder, i.e. a sequence of usually correlated random variables with mean zero. Mostly, the goal is to find a decomposition such that E_t is a stationary time series process.



There are time series, where seasonal effect and random variation increase as the trend increases. The air passenger bookings from section 1.2.1 are an example. In many of these cases, a multiplicative decomposition model is appropriate:

$$X_t = m_t \cdot s_t \cdot E_t$$

If we take logarithms, this brings us back to the additive case:

$$\log(X_t) = \log(m_t) + \log(s_t) + \log(E_t) = m_t' + s_t' + E_t'$$

For illustration, we carry out the log-transformation on the air passenger bookings; see the above. Indeed, seasonal effect and random variation now seem to be independent of the level of the series. Thus, for the original data, the multiplicative model is appropriate. However, it is now clearly evident from these logged data that the seasonal effect changes over time.

For logged series, some care is required when the exponential function is applied to the predicted mean of $\log(X_t)$ to obtain a prediction for the expectation of X_t, as the effect is usually to bias the predictions. If the process E_t is normally distributed with mean 0 and variance σ^2, then the expectation of X_t is given by:

$$E[X_t] = \hat{X}_t = \exp(m_t + s_t) \cdot \exp(\sigma^2 / 2)$$

In the following few chapters, we now explain a few methods for estimating and additive decomposition of an observed time series.

4.2.2 Differencing

A simple, yet not overly useful approach for removing deterministic trends and/or seasonal effects from a time series is by taking differences. While it is conceptually simple and quick, its main disadvantage is that it does not result in explicit estimates of trend component m_t and seasonal component s_t.

However, in the absence of a seasonal effect, a (piecewise) linear trend in a time series can be removed by taking first-order differences with lag 1:

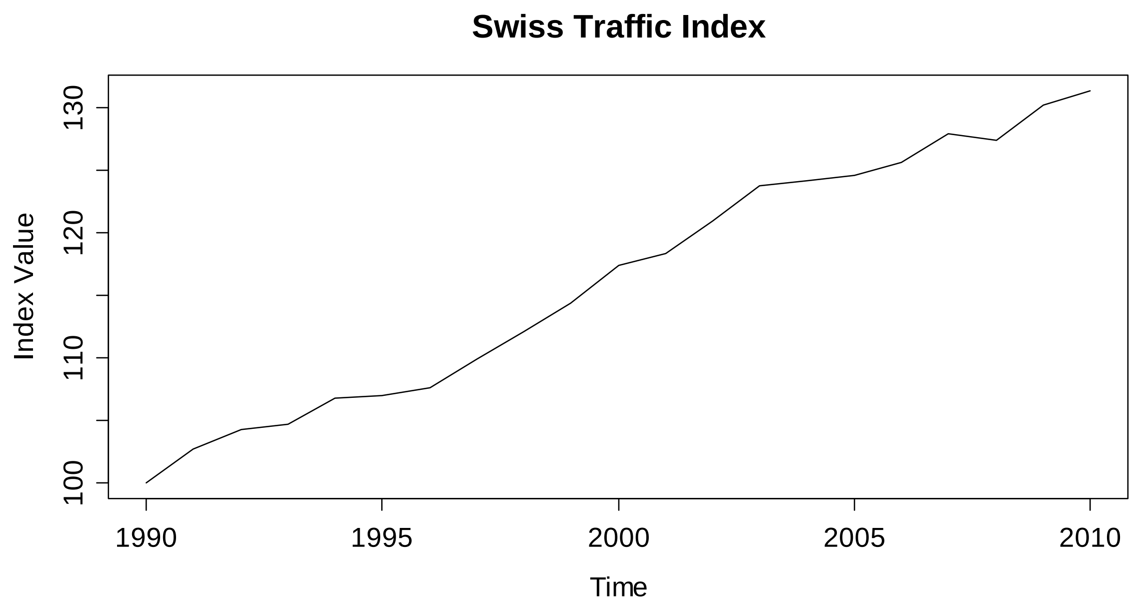
$$\begin{aligned} X_t &= \alpha + \beta t + E_t, \; E_t \; stationary \\ Y_t &= X_t - X_{t-1} = \beta + E_t - E_{t-1} \end{aligned}$$

Another somewhat disturbing property of the differencing approach is that strong, artificial new dependencies are created. Note that if E_t is a stochastically independent process, then X_t is independent, too, but the differenced process Y_t is not:

$$\begin{aligned} Cov(Y_t, Y_{t-1}) &= Cov(\beta + E_t - E_{t-1}, \beta + E_{t-1} + E_{t-2}) \\ &= -Cov(E_{t-1}, E_{t-1}) \\ &\neq 0 \end{aligned}$$

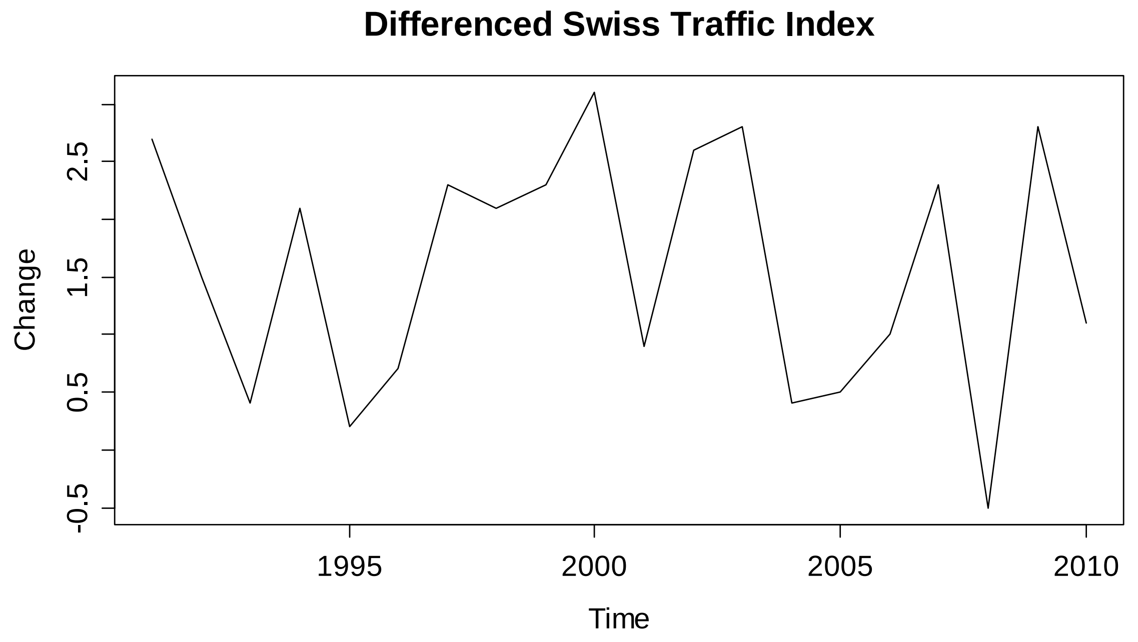
We illustrate how differencing works by using a dataset that shows the traffic development on Swiss roads. The data are available from the federal road office (ASTRA) and show the indexed traffic amount from 1990-2010. We type in the values and plot the original series:

```
> 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 at least piecewise linear. Taking first-order differences with lag 1 shows the yearly changes in the Swiss Traffic Index, this should 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 clearly has a constant mean, i.e. the trend was successfully removed.

What has differencing to offer for polynomial trends, i.e. quadratic or cubic ones? It is possible to take higher order differences to remove also these. We here show how to do it in the case of a quadratic trend.

$$\begin{aligned} X_t &= \alpha + \beta_1 t + \beta_2 t^2 + E_t,\ E_t \ \textit{stationary} \\ Y_t &= (X_t - X_{t-1}) - (X_{t-1} - X_{t-2}) \\ &= E_t - 2E_{t-1} + E_{t-2} + 2\beta_2 \end{aligned}$$

The extension to cubic trends and even higher orders is straightforward. In **R**, we can still employ function `diff()`, but have to provide argument `differences=...` for indicating the order of the difference.

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 (piecewise) linear trends, and obtain a stationary series. We take first-order differences with lag p:

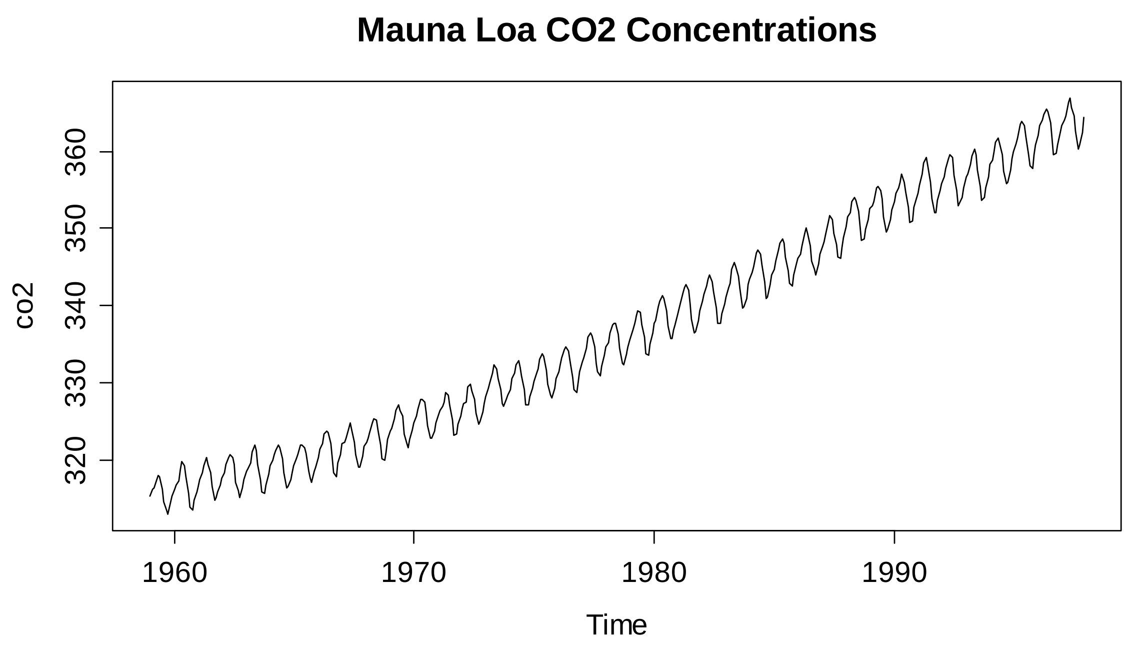
$$Y_t = X_t - X_{t-p},$$

where 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, i.e. often 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 will be removed. While taking seasonal differences still has some ability to remove only piecewise linear trends, this property is much less existent than when differencing with lag 1.

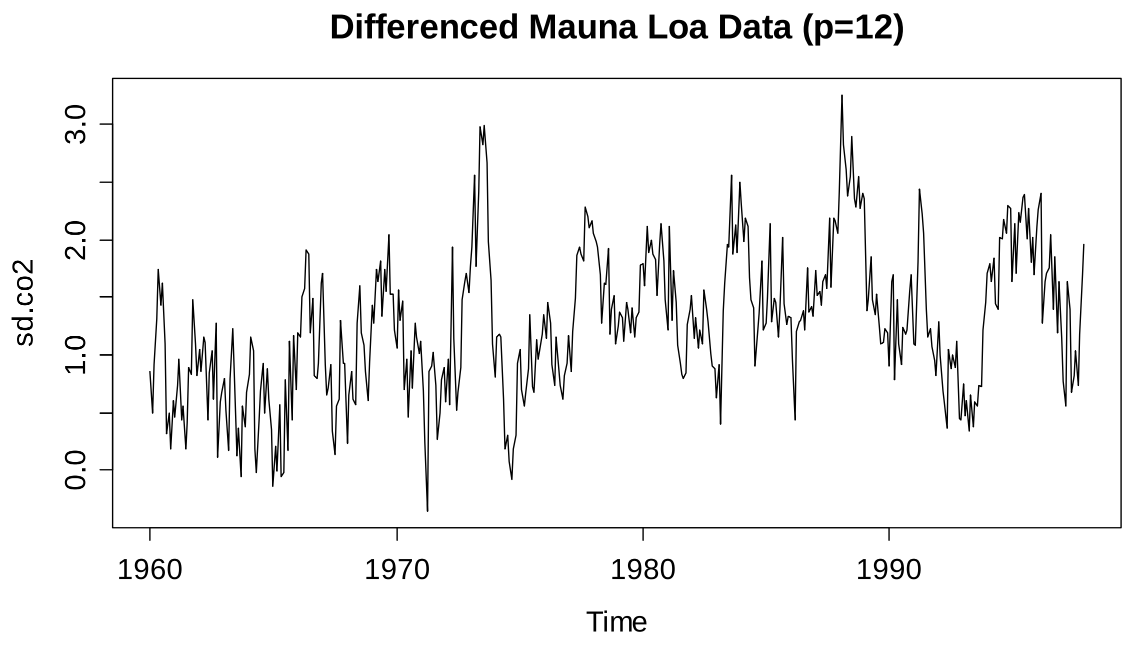
We are illustrating seasonal differencing using the Mauna Loa atmospheric CO_2 concentration data. This is a time series with monthly records from January 1959 to December 1997. It exhibits both a (almost linear) trend and a distinct seasonal pattern. We first load the data and do a time series plot:

```
> 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. It remains somewhat questionable whether it is stationary, owing to a potentially non-linear trend in the original data.



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



Because we are comparing every record with the one from the previous year, the resulting series is 12 observations shorter than the original one. We conclude this section by emphasizing again that while differencing is quick and simple, we do not obtain explicit estimates for trend m_t and seasonal effect s_t. Not surprisingly, this makes extrapolation of a series quite difficult – which of course is an issue, if one is interested in forecasting. Please note that this problem is addressed in section 7, where we discuss SARIMA models.

4.2.3 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 $\hat{E}_t$. In the absence of a seasonal effect, the trend of a time series can simply be obtained by applying an *additive linear filter*:

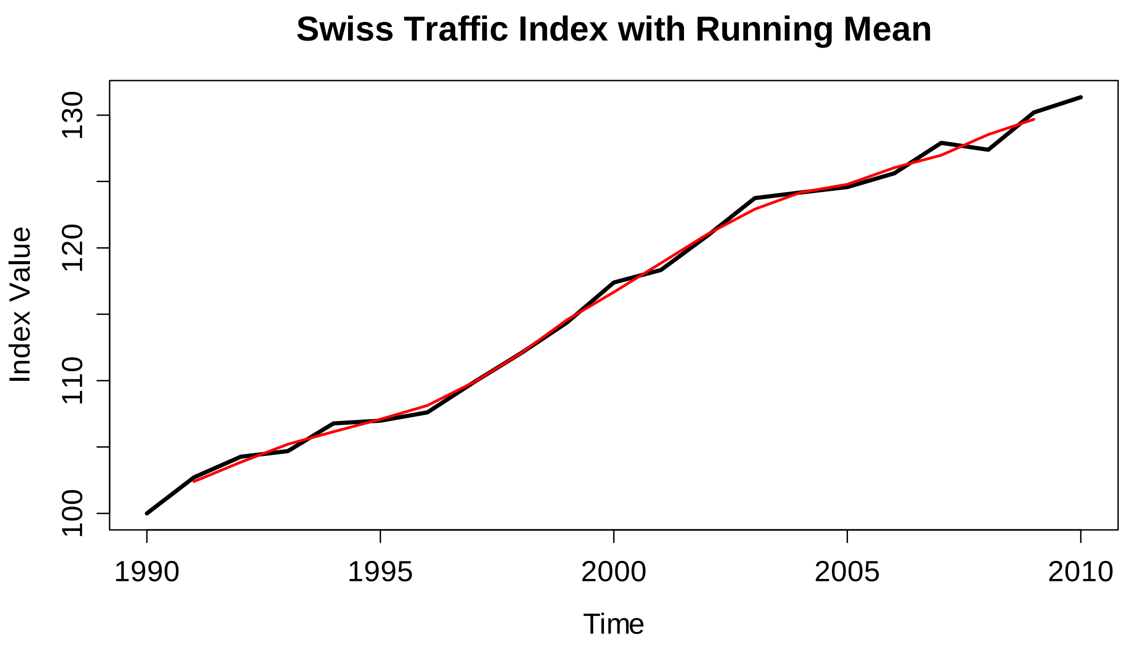
$$\hat{m}_t = \sum_{i=-p}^{q} a_i X_{t+i}$$

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

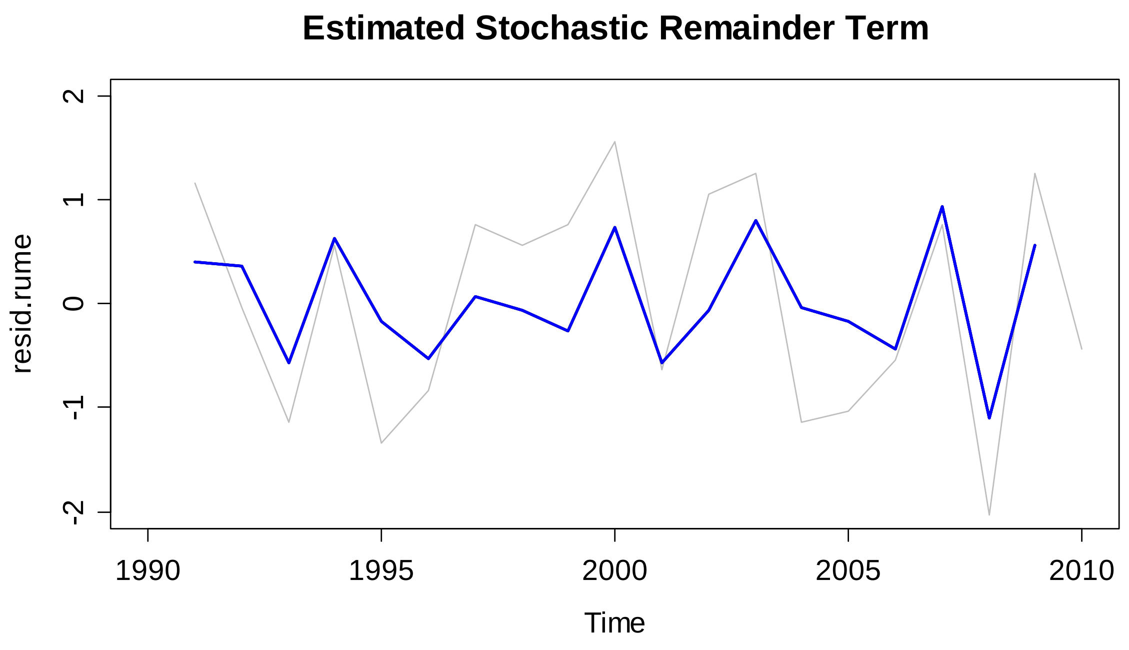
Example: Trend Estimation with Running Mean

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()` in **R**. With the correct settings, this function becomes a running mean estimator.

```
> trend.est <- filter(SwissTraffic, filter=c(1,1,1)/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. This has the consequence that for both the first and the last instance of the time series, no trend estimate is available. Also, 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 terms from differencing and running mean trend estimation:



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 both are estimates of one and the same term, i.e. the stochastic remainder.

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:

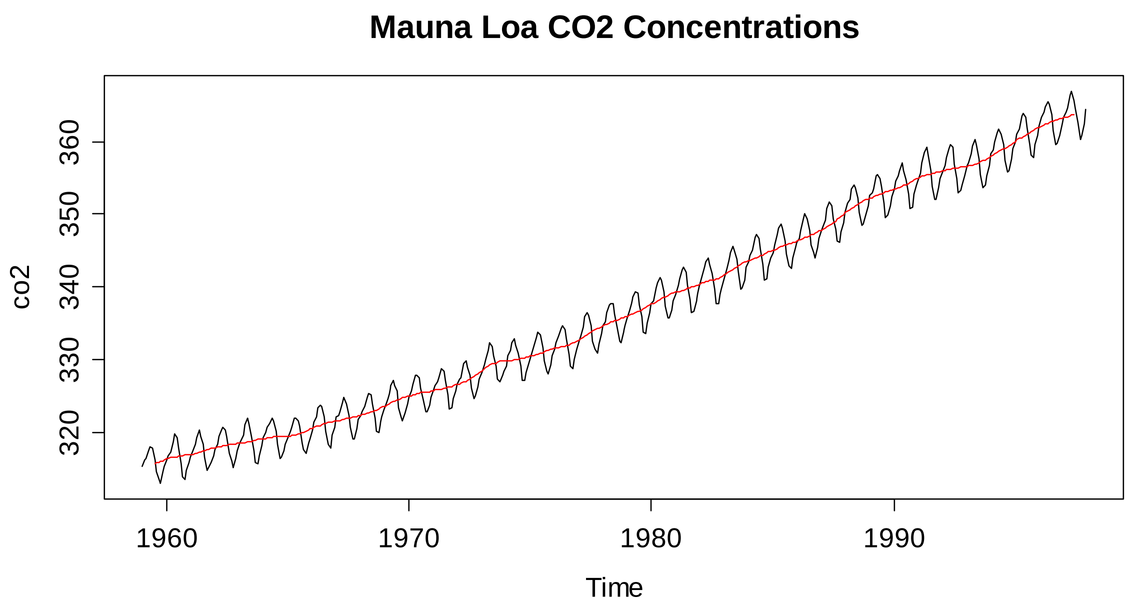
$$\hat{m}_t = \frac{1}{12}\left(\frac{1}{2}X_{t-6} + X_{t-5} + \ldots + X_{t+5} + \frac{1}{2}X_{t+6}\right), \text{ for } t = 7,...,n-6$$

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. 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)
> 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.

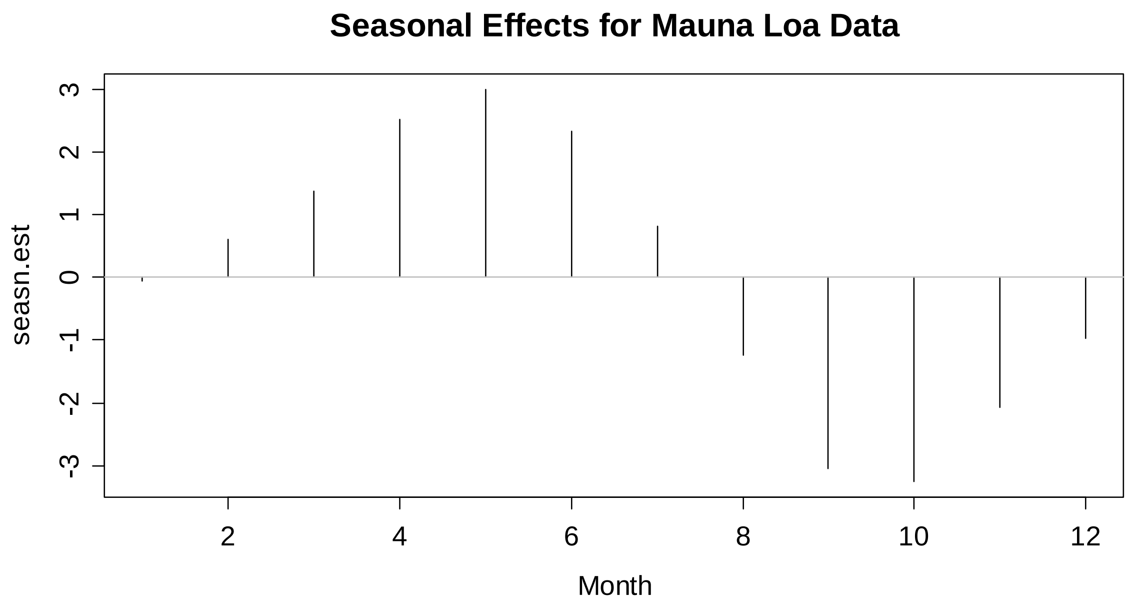
Estimation of the Seasonal Effect

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.

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

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 we need to remove in the seasonal averaging process.

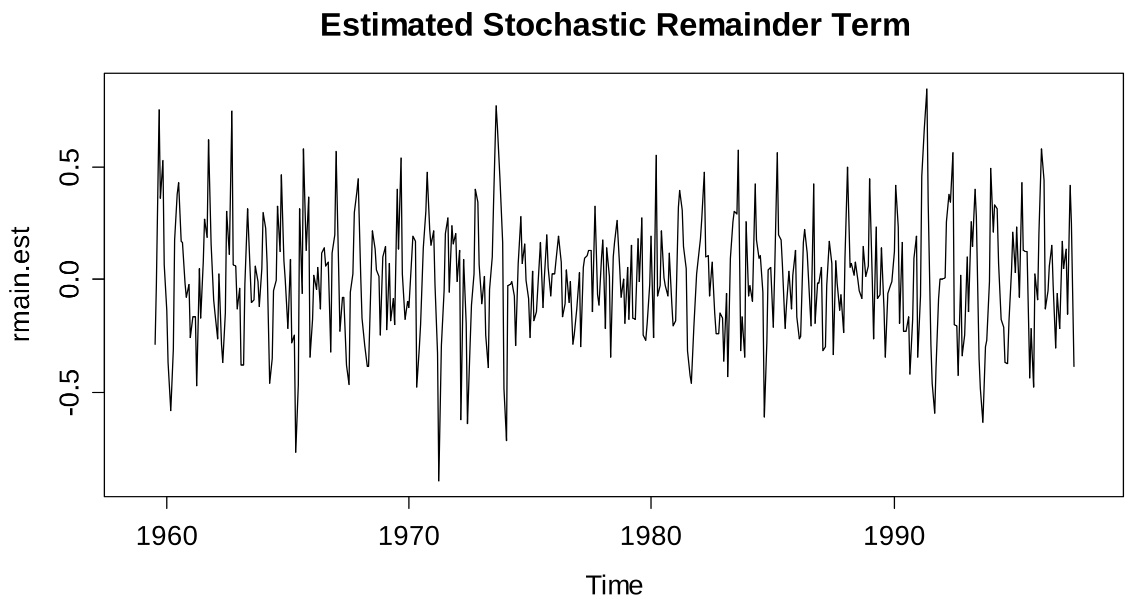
```
> trend.adj <- co2-trend.est
> month     <- factor(rep(1:12,39))
> seasn.est <- tapply(trend.adj, month, mean, na.rm=TRUE)
> plot(seasn.est, type="h", xlab="Month")
> title("Seasonal Effects for Mauna Loa Data")
> abline(h=0, col="grey")
```



In the plot above, we observe that during a season, 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

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

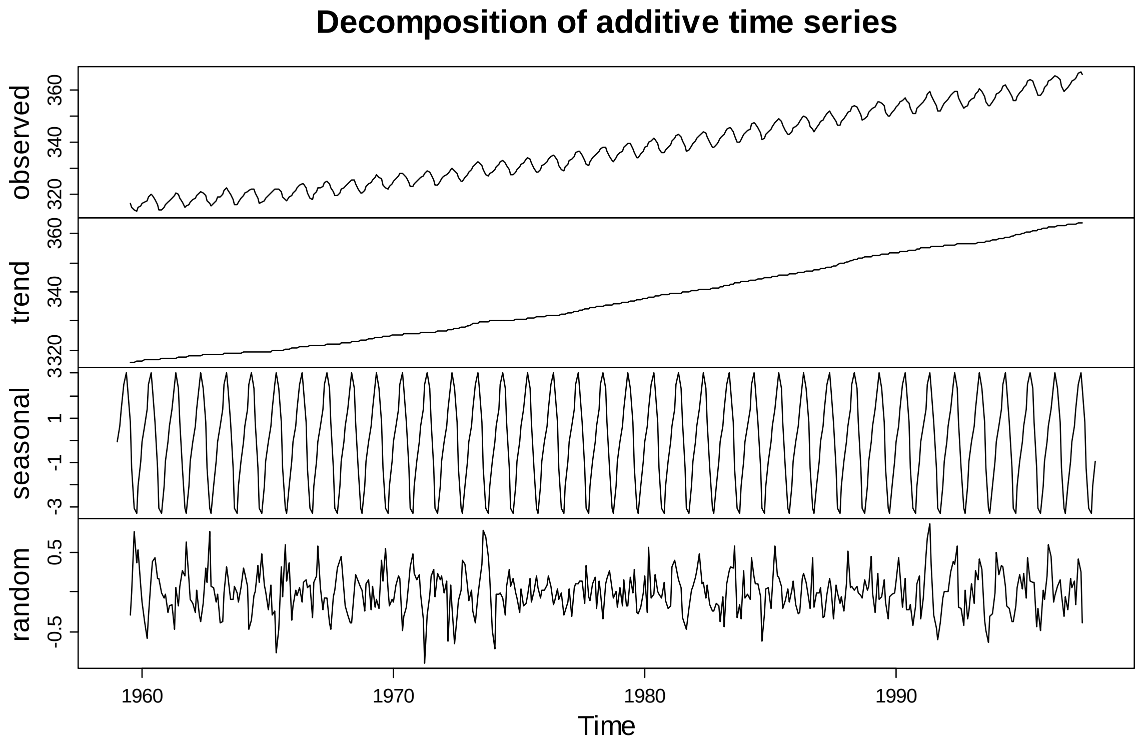
We display this below. It seems as if the remainder still has some periodicity. Does that mean that removing the seasonal effect was not successful?



The observed 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 4.2.4 about STL-decomposition.

Moreover, we would like to emphasize that R offers the convenient `decompose()` function for running mean estimation and seasonal averaging. Only for educational purposes, we had done this in a do-it-yourself manner above. 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.

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



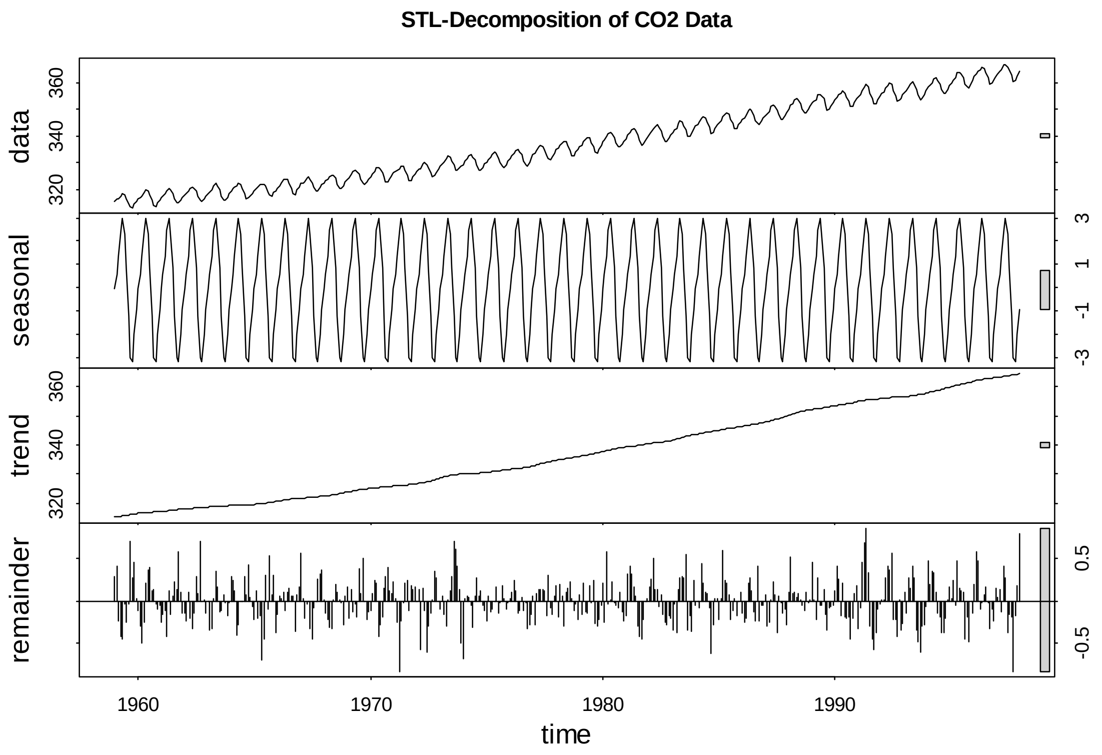
The `decompose()` function also offers a neat plotting method that shows the four frames above with the series, and the estimated trend, seasonality and remainder. Except for the different visualization, the results are exactly the same as what we had produced with our do-it-yourself approach.

4.2.4 Seasonal-Trend Decomposition with LOESS

Another algorithm in R, which offers decomposition of a time series into trend, seasonal effect and remainder, is `stl()`. The output is (nearly) equivalent to what we had obtained above with `decompose()`. However, the details behind are different, i.e. more sophisticated and complex than the simple filtering/averaging procedure which was employed so far.

Because it is beyond the scope of this applied course, we do without giving full details on the `stl`-decomposition. However, it is based on *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. In `stl()`, estimation of trend and seasonality are done iteratively. While this all sounds straightforward, the (here omitted) technical details are quite complicated.

```
> co2.stl <- stl(co2, s.window="periodic")
> plot(co2.stl, main="STL-Decomposition of CO2 Data")
```

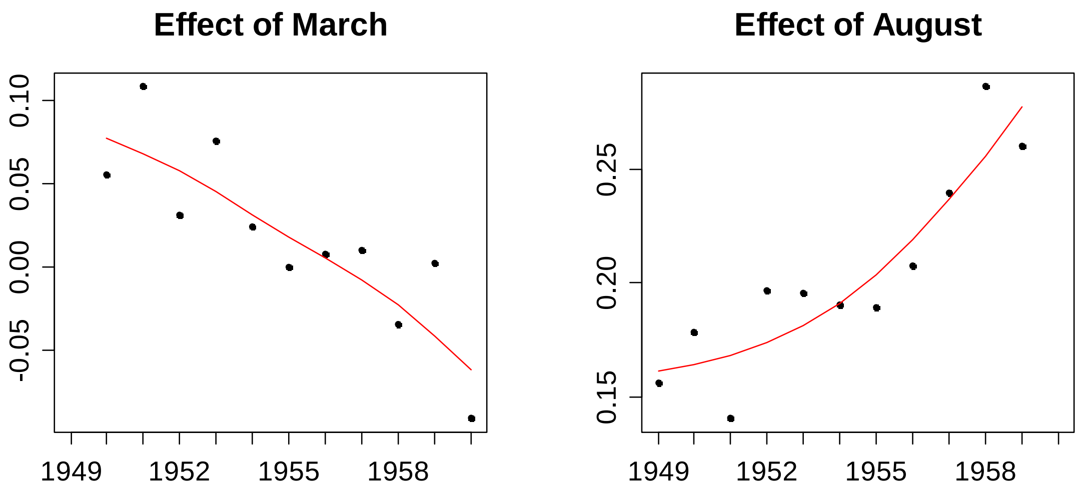


The graphical output is similar to the one on the previous page. The grey bars on the right hand side facilitate interpretation of the decomposition: they show the relative magnitude of the effects, i.e. cover the same span on the y-scale in all of the frames. The two principal arguments in function `stl()` are `t.window` and `s.window`. The first one, `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:

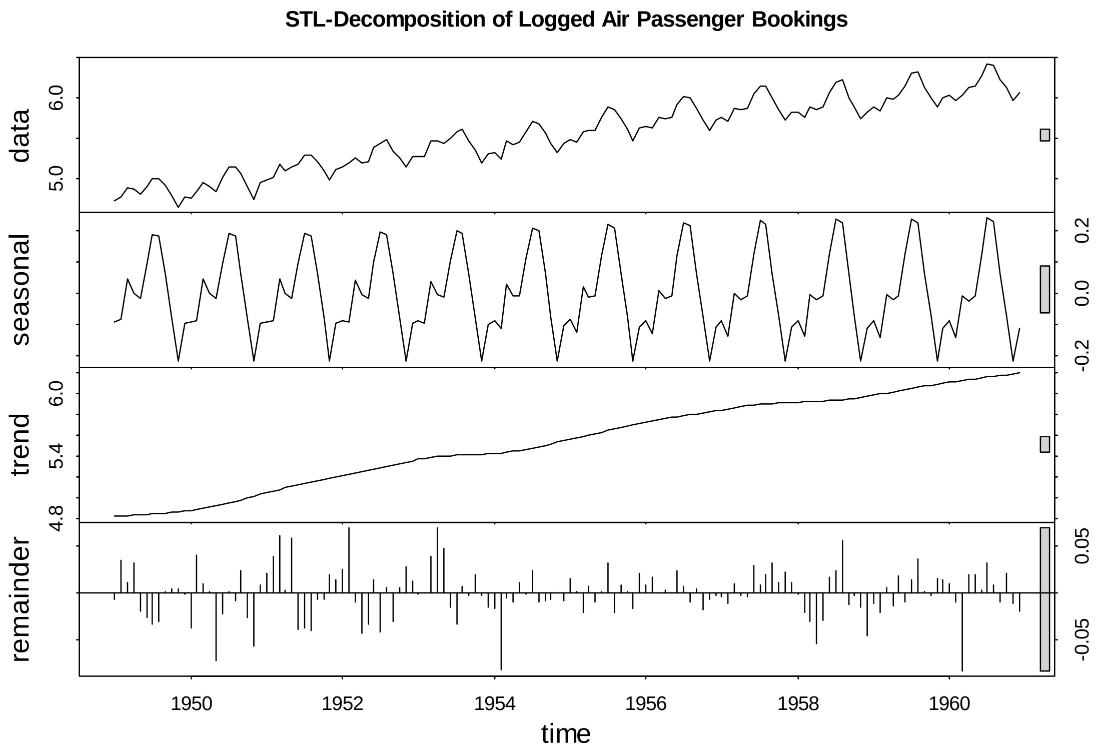
```
> co2.stl$win[2]
 t
19
```

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. The second argument, `s.window`, controls the smoothing for the seasonal effect. When set to `"periodic"` as above, the seasonality is obtained as a constant value from simple (monthly) averaging, as presented in section 4.2.3.

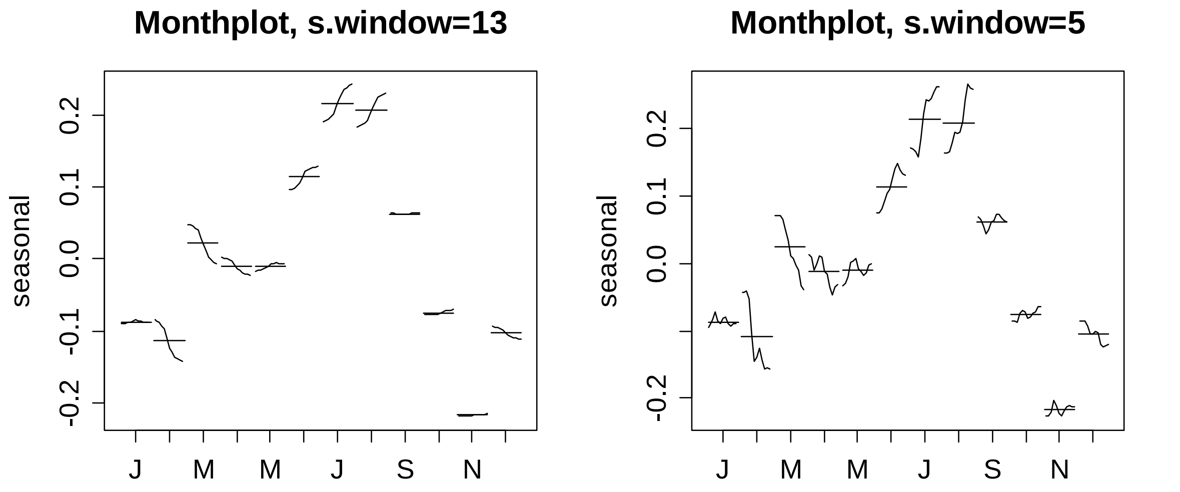
However, `stl()` offers better functionality. If `s.window` is set to a numeric value, the procedure can accommodate for evolving seasonality. The assumption behind is that the change in the seasonal effect happens slowly and smoothly. We visualize what is meant with the logged air passenger data. For quick illustration, we estimate the trend with a running mean filter, subtract it from the observed series and display all March and all August values of the trend adjusted series:



When assuming a non-changing seasonal effect, the standard procedure would be to take the mean of the data points in 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.



For obtaining a better decomposition of the air passenger bookings, we need to allow for changing seasonal effect. We achieve this by employing the `stl()` function and setting `s.window=13`. The resulting graphical output is displayed on the previous page. Please note 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()`.



On the left, we observe appropriate smoothing. However on the right, with smaller span, we observe overfitting – the seasonal effects do not evolve in a smooth way, and it means that this is not a good decomposition estimate.

4.2.5 Parametric Modeling

A powerful approach for decomposing time series is parametric modeling. It is based on the assumption of a functional form for the trend, usually a polynomial. For the seasonal effect, we can either use the dummy variable approach that amounts to averaging. Or, in some special cases, a sine/cosine seasonality may be appropriate. We illustrate the parametric modeling approach by two examples and use them for discussing some specifics.

We consider the Maine unemployment data from section 4.1.1. Our goal is to fit a polynomial trend, along with a seasonal effect that is obtained by averaging. We write down this model for a polynomial of grade 4.

$$X_t = \beta_0 + \beta_1 \cdot t + \beta_2 \cdot t^2 + \beta_3 \cdot t^3 + \beta_4 \cdot t^4 + \alpha_{i\langle t \rangle} + E_t, ,$$

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 what polynomial order is appropriate, and how this model can be fit.

As for the fitting, this will be done with the least squares algorithm. This requires some prudence, because we assume a remainder term E_t which is not necessarily stochastically independent. Thus, we have some violated assumption for the ordinary least squares (OLS) estimation. Since the estimated coefficients are still unbiased, OLS is a valid approach. However, be careful with the standard errors, as well as tests and confidence intervals derived from them, because they can be grossly misleading.

For the grade of the polynomial, we determine by eyeballing from the time series plot that the hypothesized trend in the unemployment series has at least 3 minima. This means that a polynomial with grade below 4 will not result in a sensible trend estimate. Thus, we try orders 4, 5 and 6, and discuss how an appropriate choice can be made from residual analysis. However, we first focus on the R code for fitting such models:

```
> maine <- ts(dat, start=c(1996,1), freq=12)
> tr    <- as.numeric(time(maine))
> tc    <- tr-mean(tr)
> mm    <- rep(c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                 "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"))
> mm    <- factor(rep(mm,11),levels=mm)[1:128]
```

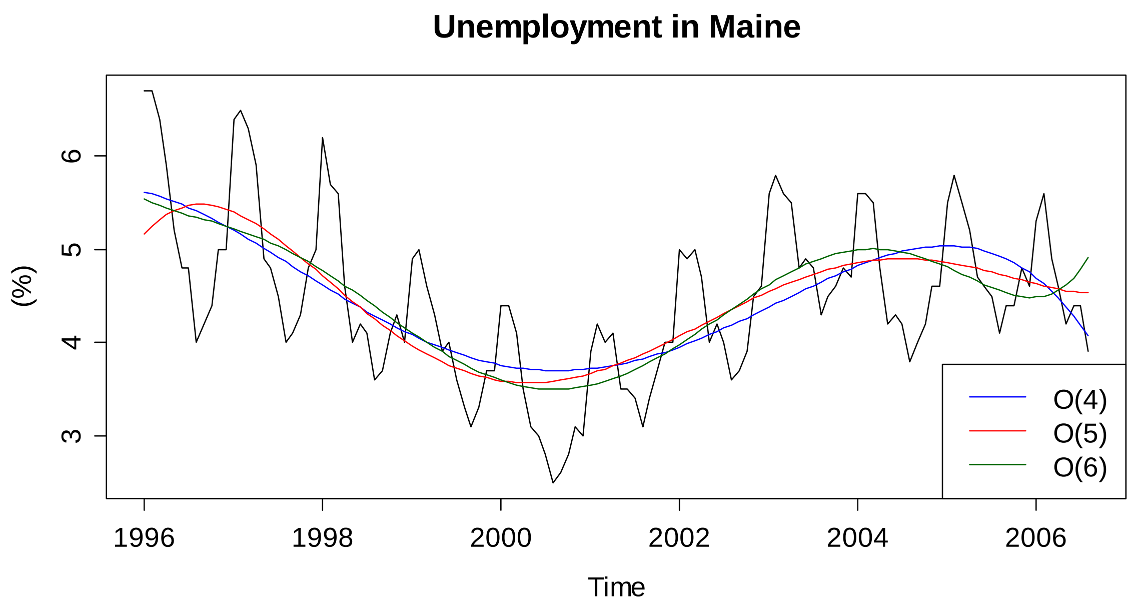
In a first step, we lay the basics. From time series `maine`, we extract the times of observation as the predictor. As always when fitting polynomial regression models, it is crucial to center the x-values to mitigate potential collinearity among the terms. Furthermore, we define a factor variable for modeling the seasonality.

```
> fit04    <- lm(maine~tc+I(tc^2)+I(tc^3)+I(tc^4)+mm)
> cf       <- coef(fit04)
> t.est.04 <- cf[1]+cf[2]*tc+cf[3]*tc^2+cf[4]*tc^3+cf[5]*tc^4
> t04.adj  <- t.est.04-mean(t.est.04)+mean(maine)
```

We can obtain an OLS-fit of the decomposition model with **R**'s `lm()` procedure. The `I()` notation in the formula assures that the "^" are interpreted as arithmetical operators, i.e. powers of the predictor, rather than as formula operators. Thereafter, we can use the estimated coefficients for determining the trend estimate `t.est.04`. Because the seasonal factor uses the month of January as a reference, and thus generally has a mean different from zero, we need to shift the trend to make run through "the middle of the data" – this is key if we aim for visualizing the trend.

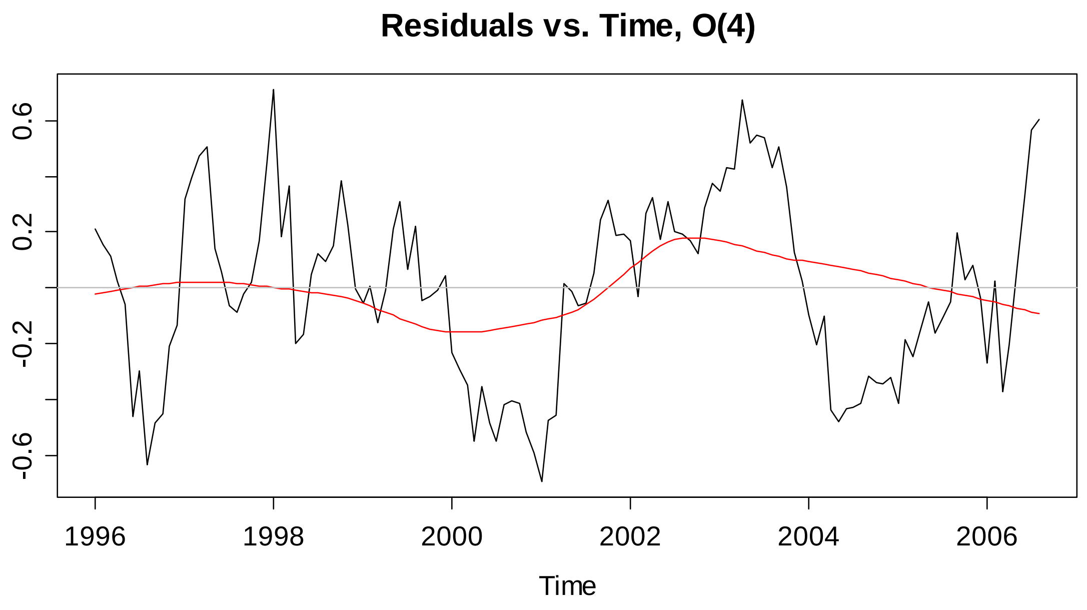
```
> plot(maine, ylab="(%)", main="Unemployment in Maine")
> lines(tr, t.04.adj)
```

The time series plot on the next page is enhanced with polynomial trend lines of order 4 (blue), 5 (red) and 6 (green). From this visualization, it is hard to decide which of the polynomials is most appropriate as a trend estimate. Because there are some boundary effects for orders 5 and 6, we might guess that their additional flexibility is not required. As we will see below, this is treacherous.

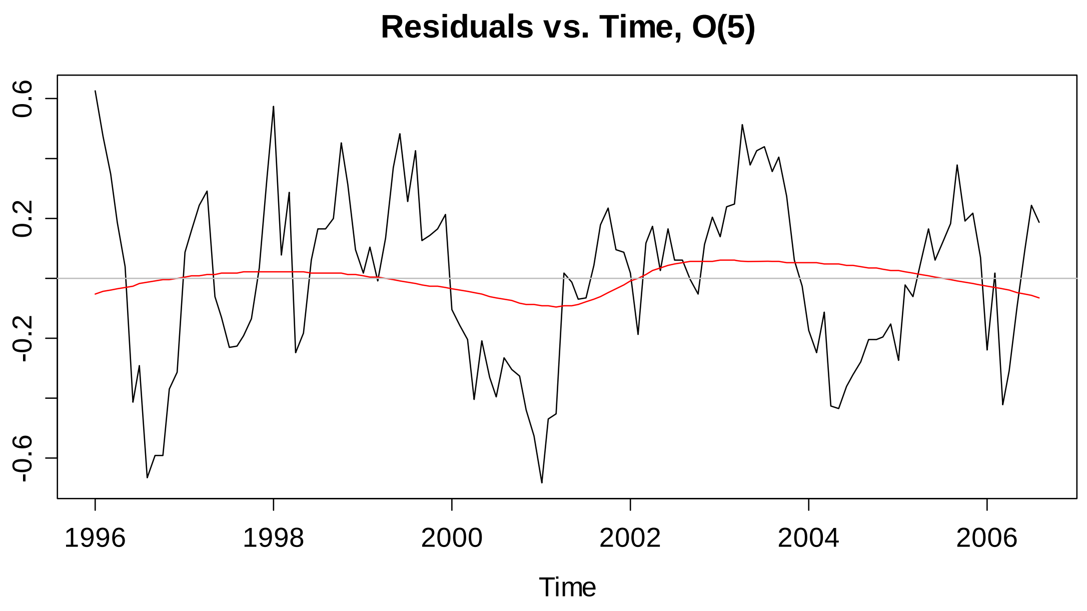


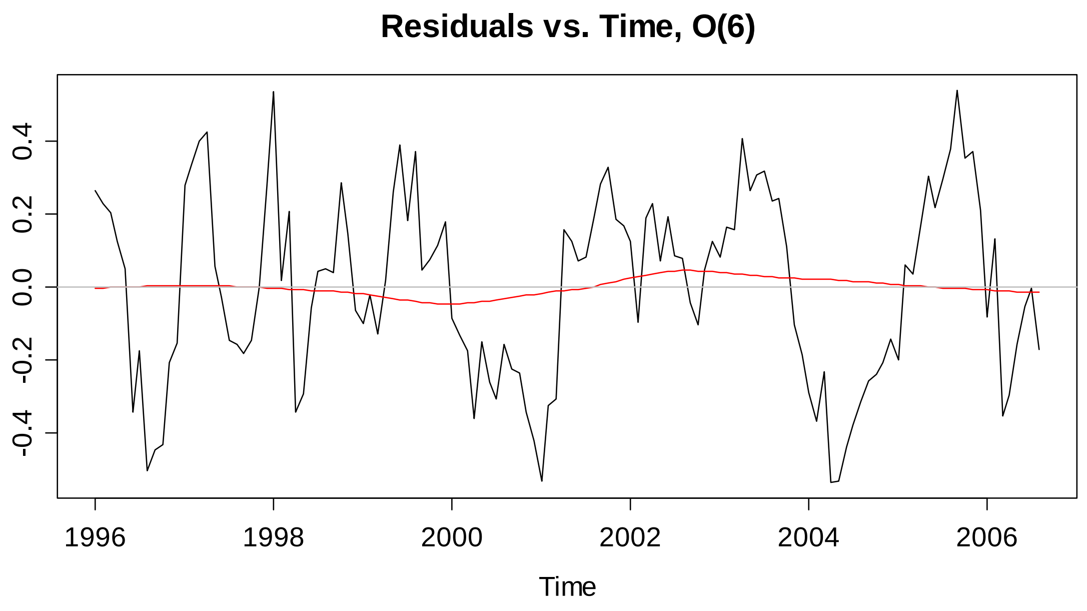
A better way for judging the fit of a parametric model is by residual analysis. We plot the remainder term $\hat{E}_t$ versus time and add a LOESS smoother.

```
> re.est <- maine-fitted(fit04)
> plot(re.est, ylab="", main="Residuals vs. Time, O(4)")
> fit <- loess(re.est~tr)
> lines(tr, fitted(fit), col="red")
> abline(h=0, col="grey")
```



The above plot shows some, but not severe, lack of fit, i.e. the remainder term still seems to have a slight trend, owing to a too low polynomial grade. The picture becomes clearer when we produce the equivalent plots for grade 5 and 6 polynomials. These are displayed on the next page.





The residuals look best in the last plot for order 6, which would be the method of choice for this series. It is also striking that the remainder is not an i.i.d. series, the serial correlation is clearly standing out. In the next section, we will address the estimation and visualization of such autocorrelations.

We conclude this chapter on parametric modeling by issuing a warning: while the explicit form of the trend can be useful, it shall never be interpreted as causal for the evolvement of the series. Also, much care needs to be taken if forecasting is the goal. Extrapolating high-order polynomials beyond the range of observed times can yield very poor results. We will discuss some simple methods for trend extrapolation later in section 8 about forecasting.

4.3 Autocorrelation

An important feature of time series is their 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:

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

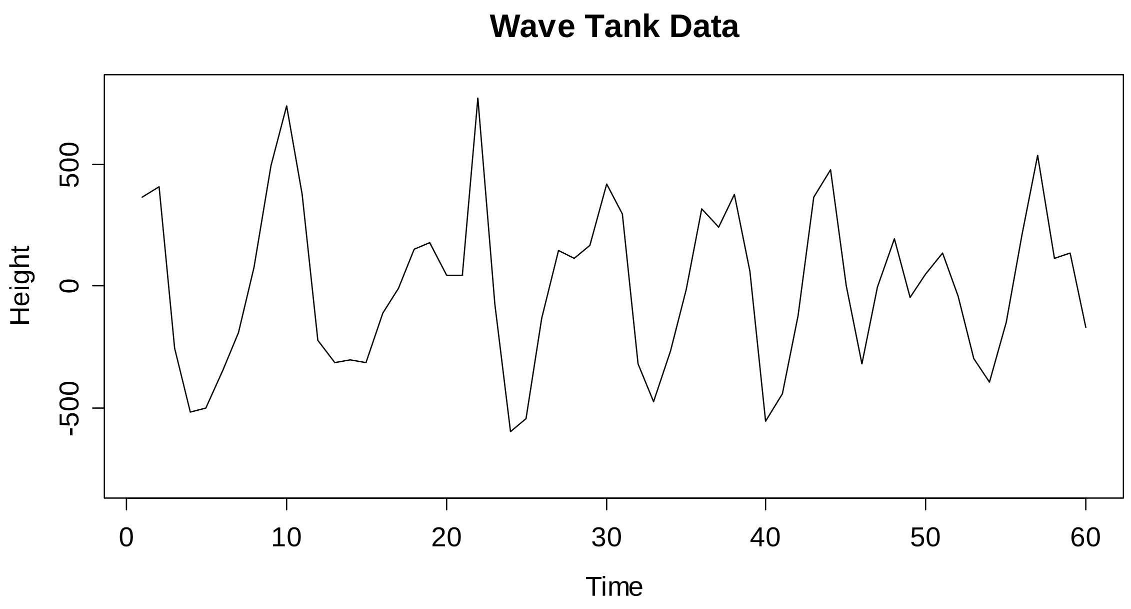
This is a dimensionless measure for the linear association between the two random variables. Since for stationary series, we require the moments to be non-changing over time, we can drop the index t for these, and write the autocorrelation as a function of the lag k:

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

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://www.massey.ac.nz/~pscowper/ts/wave.dat"
> wave <- ts(read.table(www, header=TRUE)$waveht)
> plot(window(wave, 1, 60), ylim=c(-800,800), ylab="Height")
> title("Wave Tank Data")
```

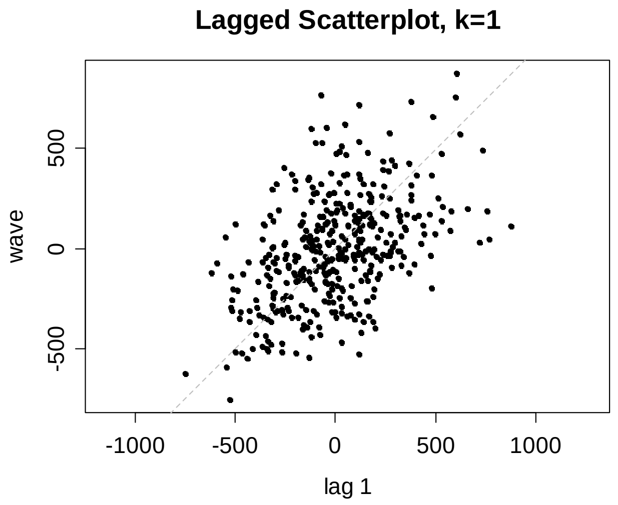


These data show some pronounced cyclic behavior. This does not come surprising, as we all know from personal experience that waves do appear in cycles. The series shows some very clear serial dependence, because the current value is quite closely linked to the previous and following ones. But very clearly,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

4.3.1 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")
```



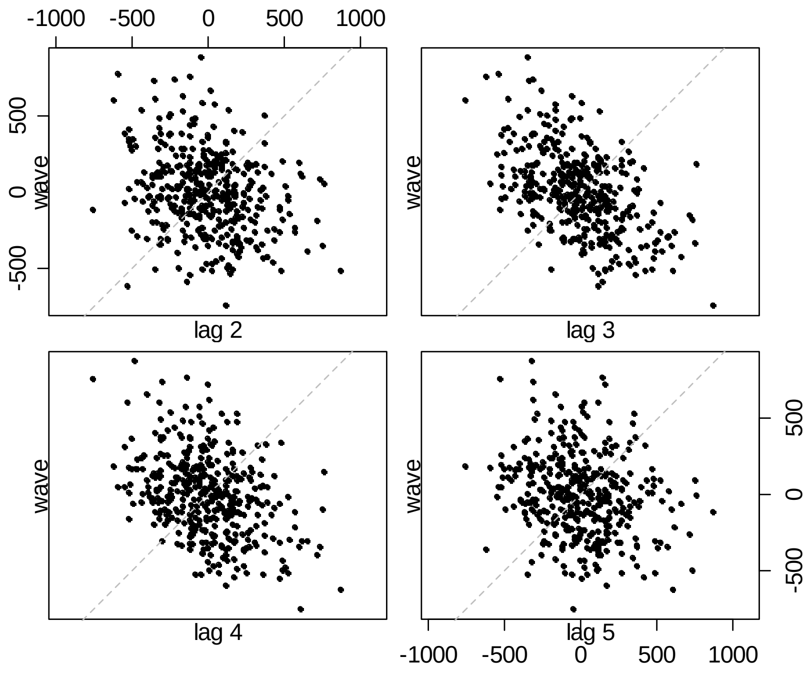
The association seems linear and is positive. The Pearson correlation coefficient turns out to be 0.47, thus moderately strong. How to interpret this value from a practical viewpoint? Well, the square of the correlation coefficient, $0.47^2 = 0.22$, is the percentage of variability explained by the linear 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 below. 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.

4.3.2 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:

$\hat{\rho}(k) = \frac{\hat{\gamma}(k)}{\hat{\gamma}(0)}$, 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:

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

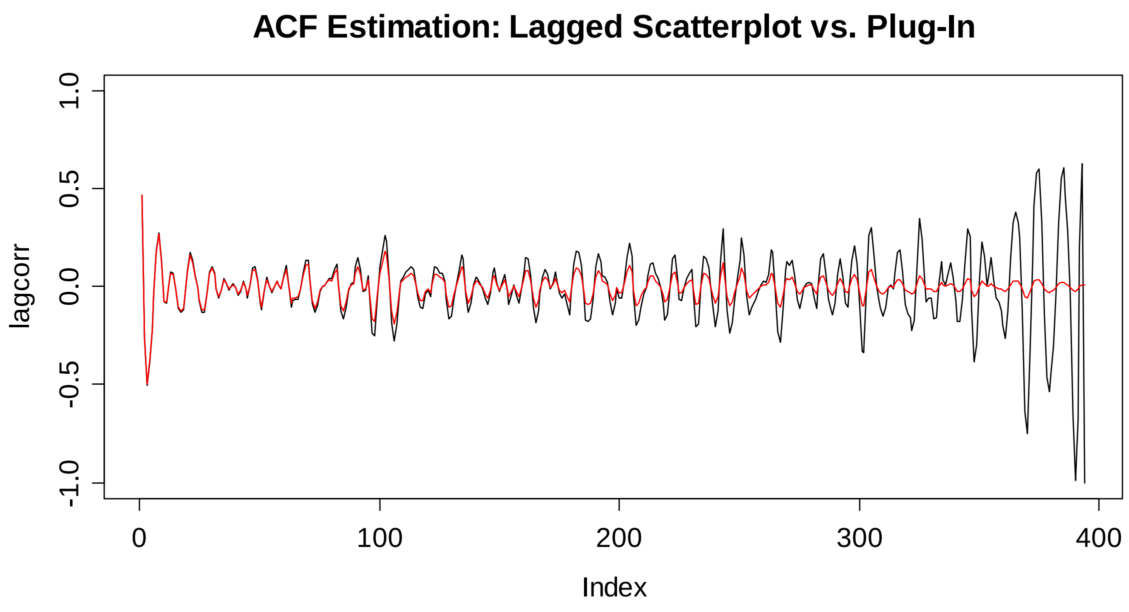
It is straightforward to compute these in **R**, function `acf()` does the job, and we below do so for the wave tank data. As for the moment, we are interested in the numerical results, we set argument `plot=FALSE`. However, as we will see below, it is usually better to visualize the estimated autocorrelation coefficients graphically, as it will be explained below in section 4.3.3. 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
```

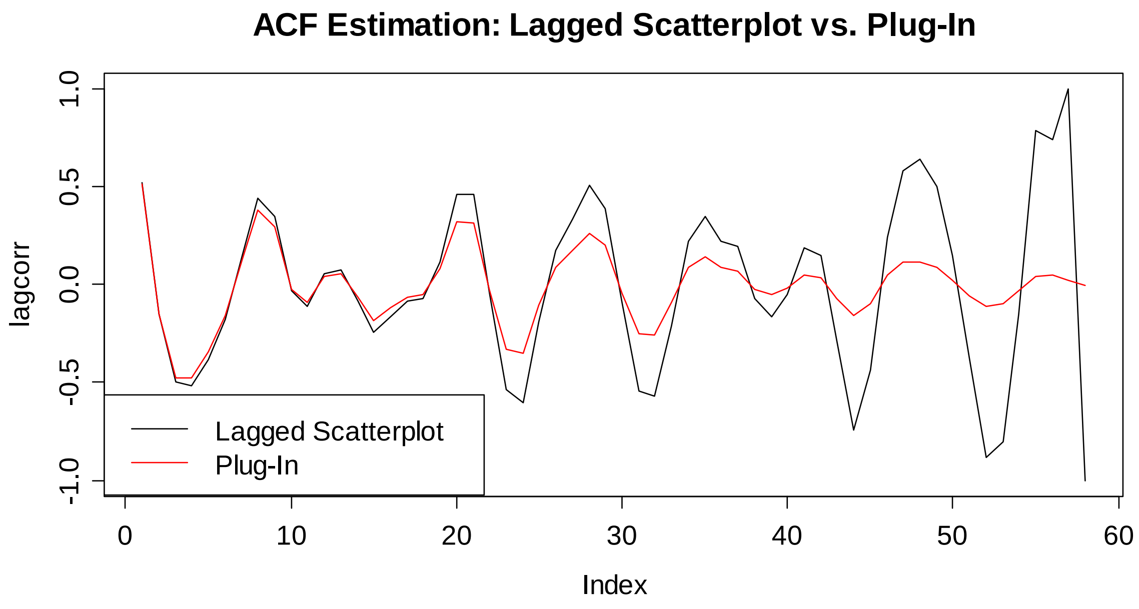
Next, we compare the autocorrelations from lagged scatterplot estimation vs. the ones from the plug-in approach. These are displayed on the next page. While for the first 50 lags, there is not much of a difference, the plug-in estimates are much more damped for higher lags. As claimed above, the lagged 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 only if we restrict the series to the first 60 observations and then repeat the estimation of autocorrelations. Again, for the highest few legs 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.



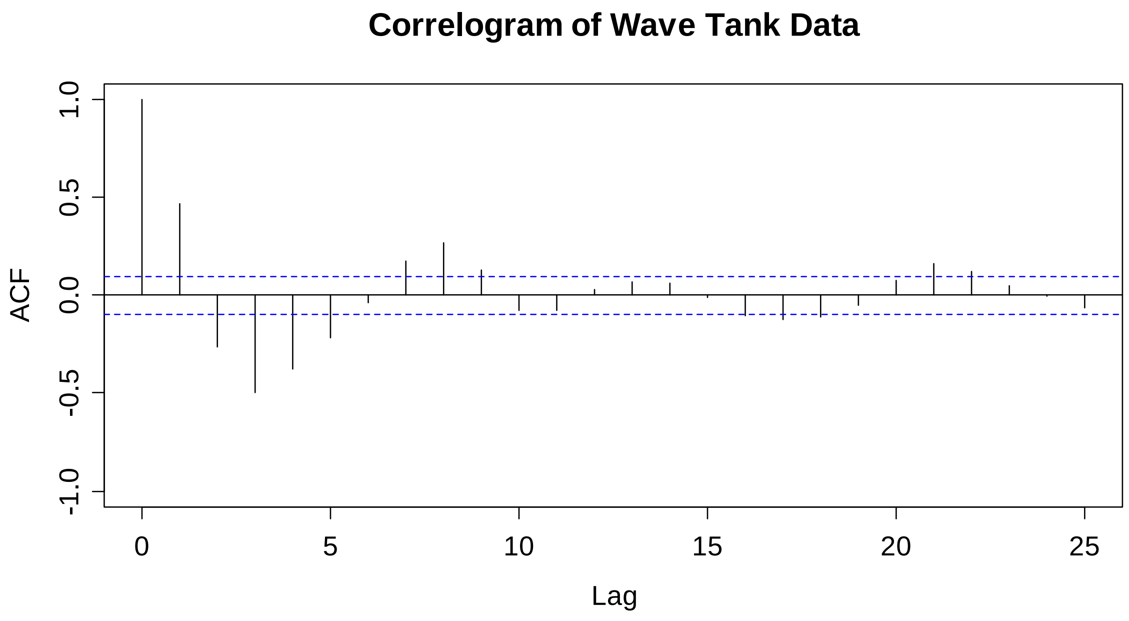
Nevertheless, all our findings still suggest that it is a good idea to consider only a first portion of the estimated autocorrelations. A rule of the thumb suggests that $10 \cdot \log_{10}(n)$ is a good threshold. For a series with 100 observations, the threshold becomes lag 20. A second rule operates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



4.3.3 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 obtain:

```
> 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 starts with lag 0, which always takes the value 1. 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 close, but the question is how close. An answer is indicated by the confidence bands, i.e. the blue dashed lines in the plot above.

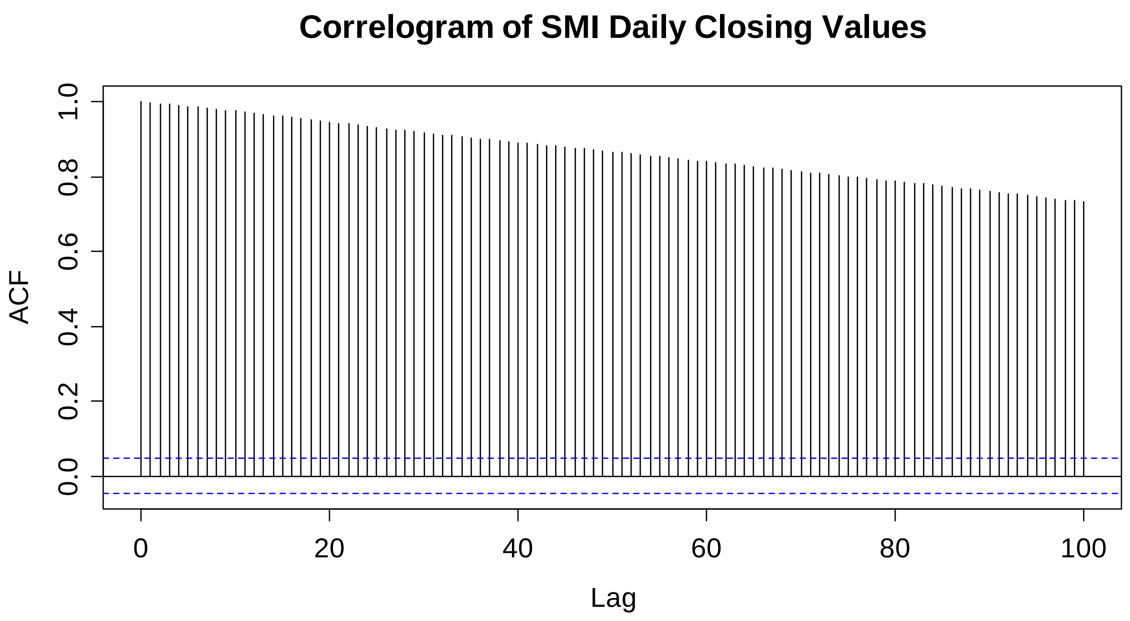
These so-called confidence bands are obtained from an asymptotic result: for long iid time series it can be shown that the $\hat{\rho}(k)$ approximately follow a $N(0,1/n)$ distribution. Thus, each $\rho(k)$ lies within the interval of $\pm 1.96/\sqrt{n}$ with a probability of approximately 95%. This 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 $\pm 2/\sqrt{n}$ 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. Please note again that the indicated bounds are asymptotic and derived from iid series. The properties of serially dependent series are much harder to derive.

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)
```

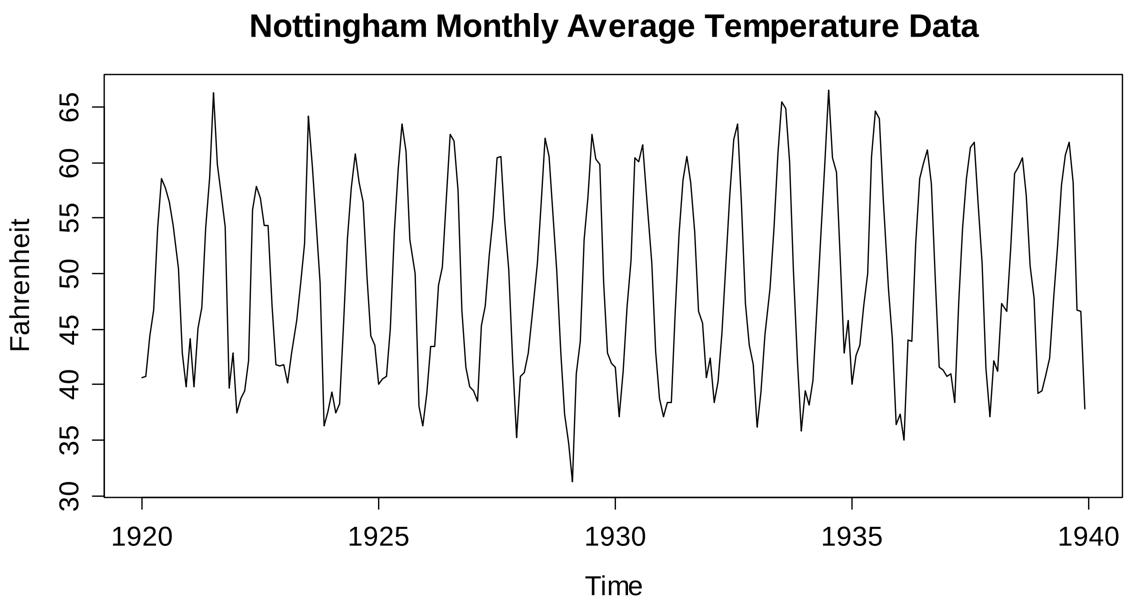


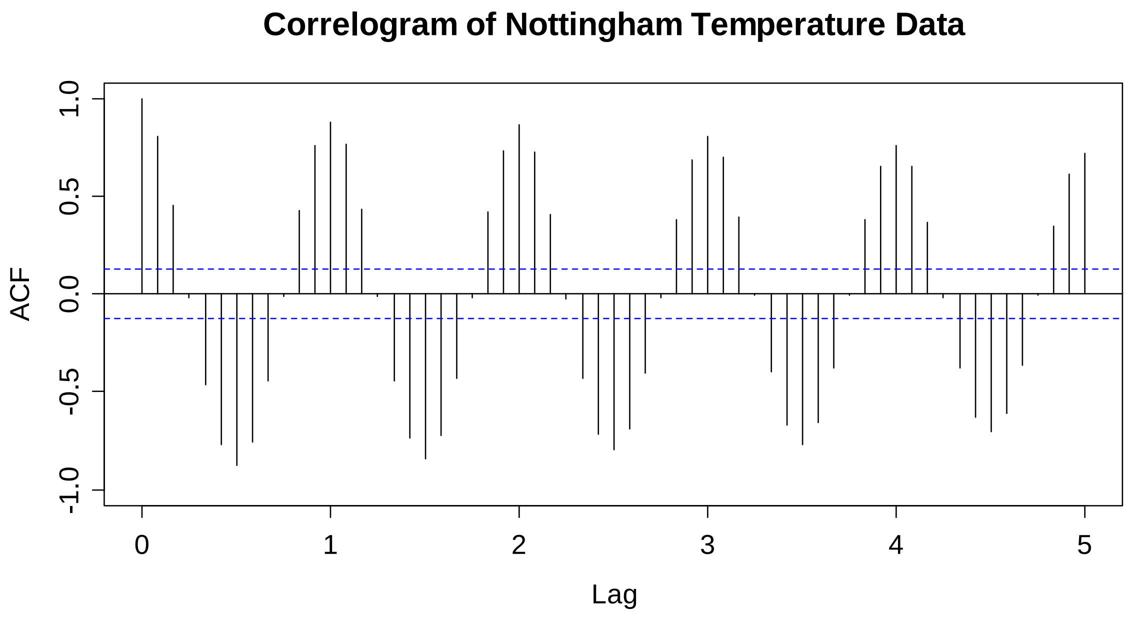
We observe that the ACF decays very slowly. The reason is that if a time series features a trend, the observations at consecutive observations will usually be on the same side of the series' global mean $\bar{x}$. This is why that for small to moderate lags k, most of the terms

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

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.

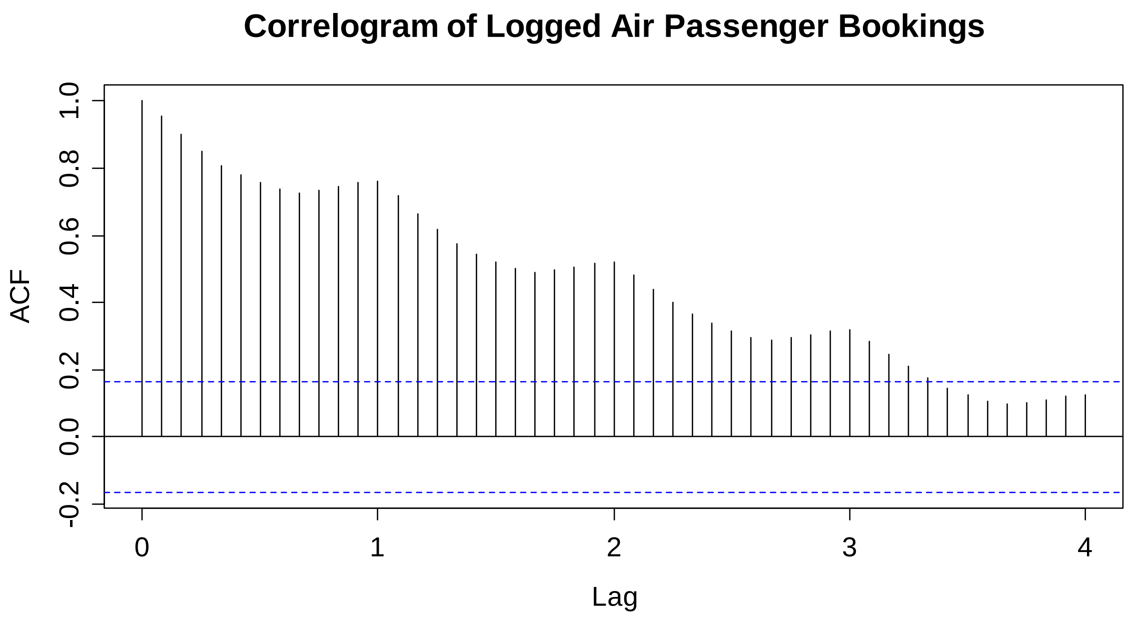
Next, we show an example of a series that has no trend, but a strongly recurring seasonal effect. We use **R**'s `data(nottem)`, a time series containing monthly average air temperatures at Nottingham Castle in England from 1920-1939. Time series plot and correlogram are as follows:





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)
```



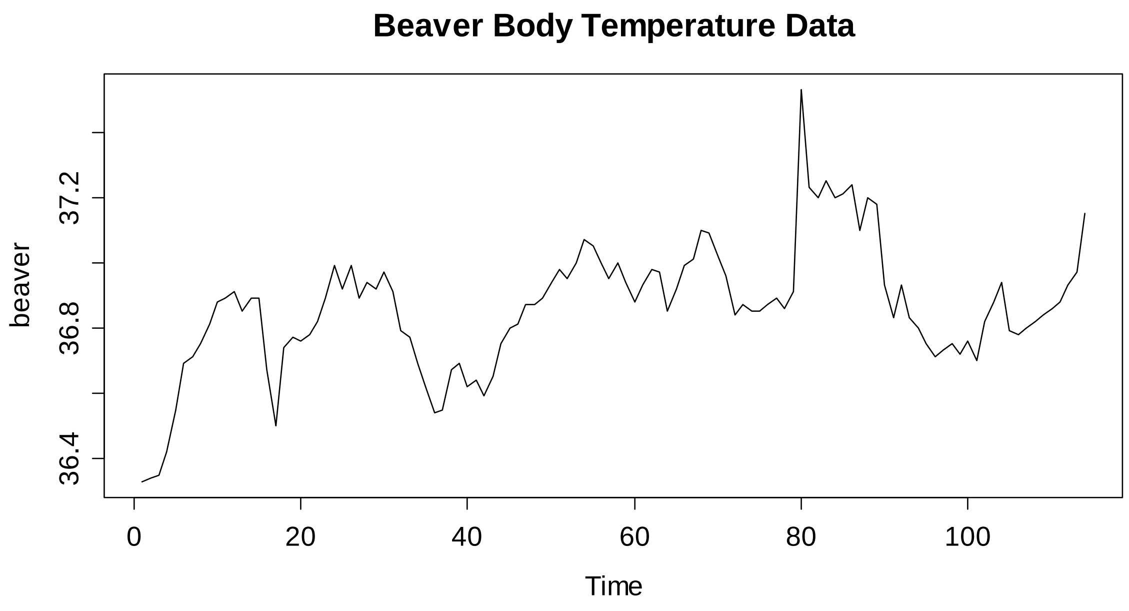
Here, the two effects described above are interspersed. We have a (here dominating) slow decay in the general level of the ACF, plus some periodicity. Again, this is an indication for a non-stationary series. It needs to be decomposed, before the serial correlation in the stationary remainder term can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

The ACF and Outliers

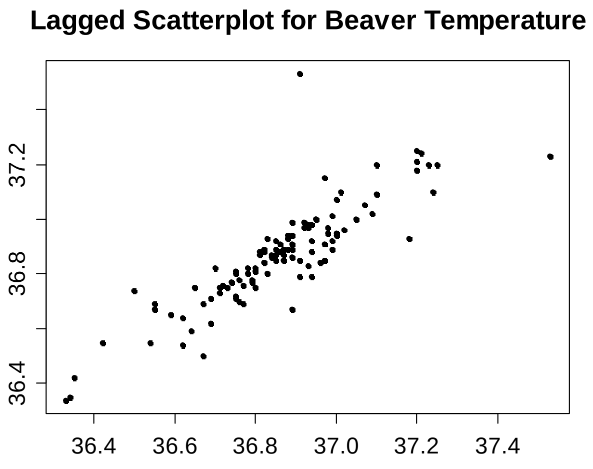
If a time series has an outlier, it will appear twice in any lagged scatterplot, and will thus potentially have “double” negative influence on the $\hat{\rho}(k)$. As an example, we consider variable `temp` from data frame `beaver1`, which can be found in **R**’s `data(beavers)`. This is the body temperature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

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. However, the purpose of this section is showing the potential bad influence of erroneous values, so we do not bother too much. Because the Pearson correlation coefficient, as well as the plug-in autocorrelation estimator is clearly non-robust, the appearance of the correlogram can be altered quite strongly due to the presence of just one single outlier.

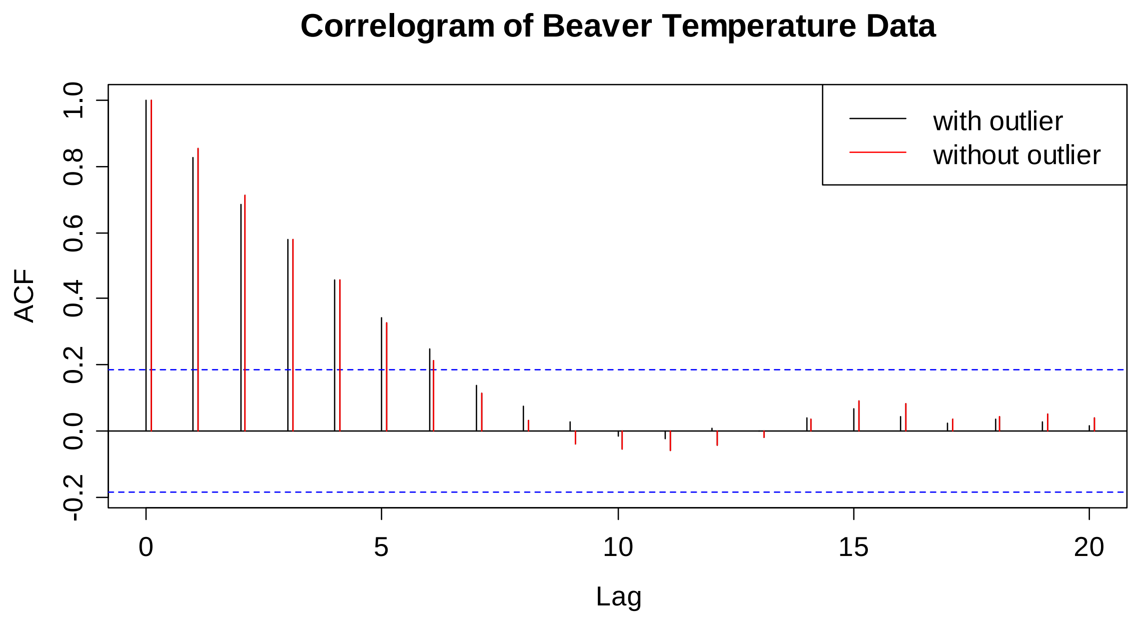
```
> data(beavers)
> beaver <- ts(beaver1$temp, start=1, freq=1)
> plot(beaver, main="Beaver Body Temperature Data")
```



```
> 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 easily identifiable. We compute the Pearson correlation coefficients with and without these observations; they are 0.86 and 0.91, respectively. Depending on how severe the outlier is, the effect can be much stronger of course. On the next page, we also show 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.

4.3.4 Quality of ACF Estimates

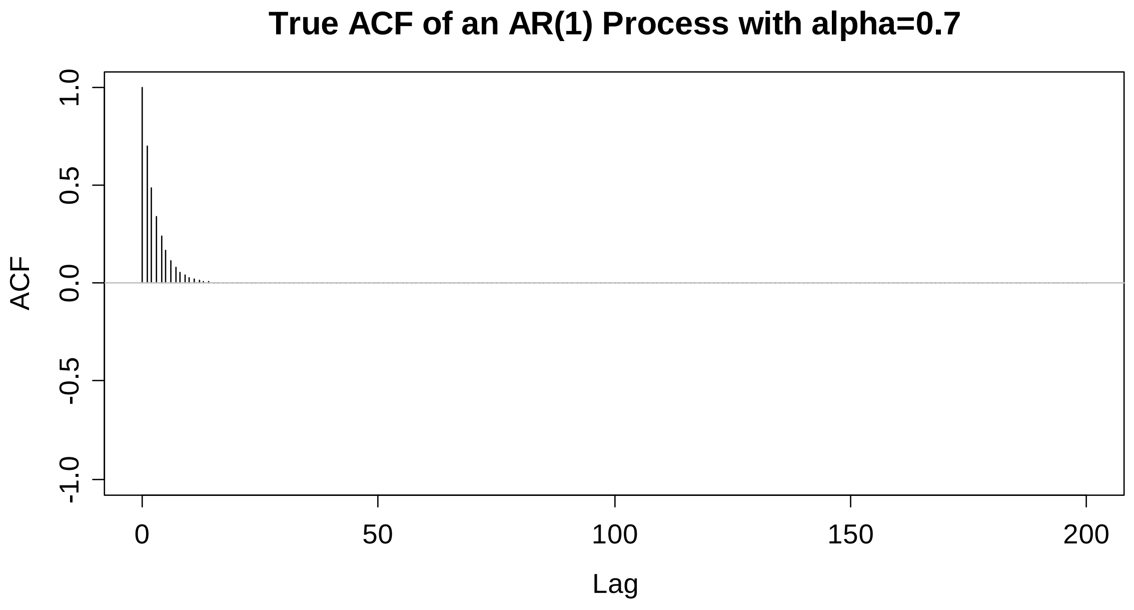
In this section we will deal with the quality of the information that is contained in the correlogram. We will not do this from a very theoretical viewpoint, but rather focus on the practical aspects. We have already learned that the ACF estimates are generally biased, i.e. damped for higher lags. This 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 is even more...

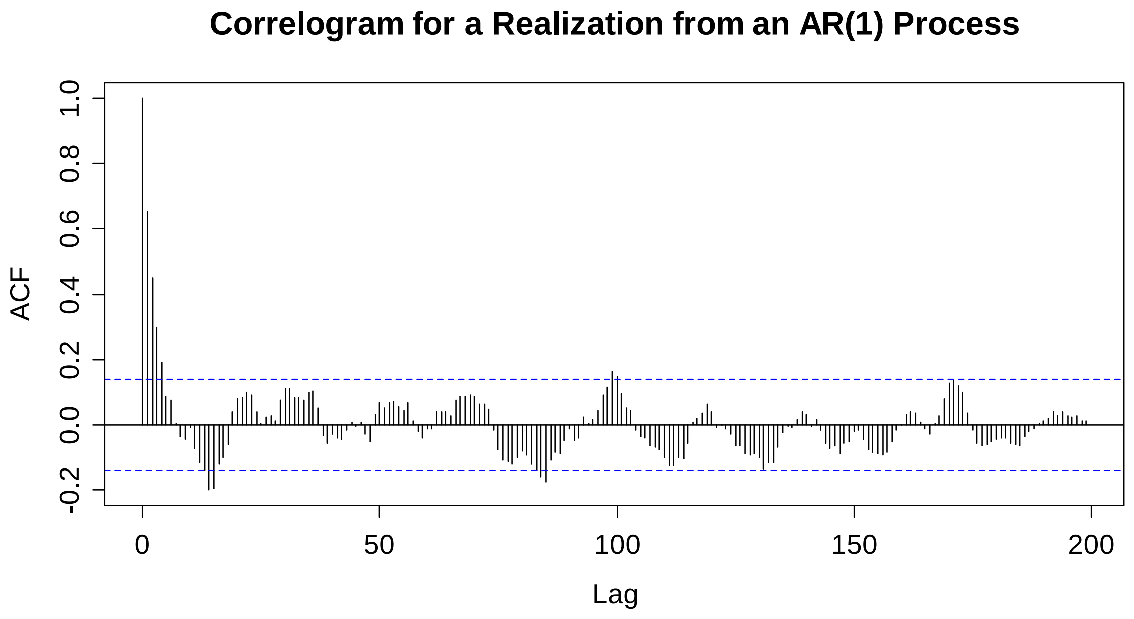
The Compensation Issue

One can show that the sum of all autocorrelations which can be estimated from a series realization is -1/2. Or, written as a formula:

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

We omit the proof here. 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 $\rho(k) > 0$ for all k, and the sample correlogram for a realization of that process with 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. 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.

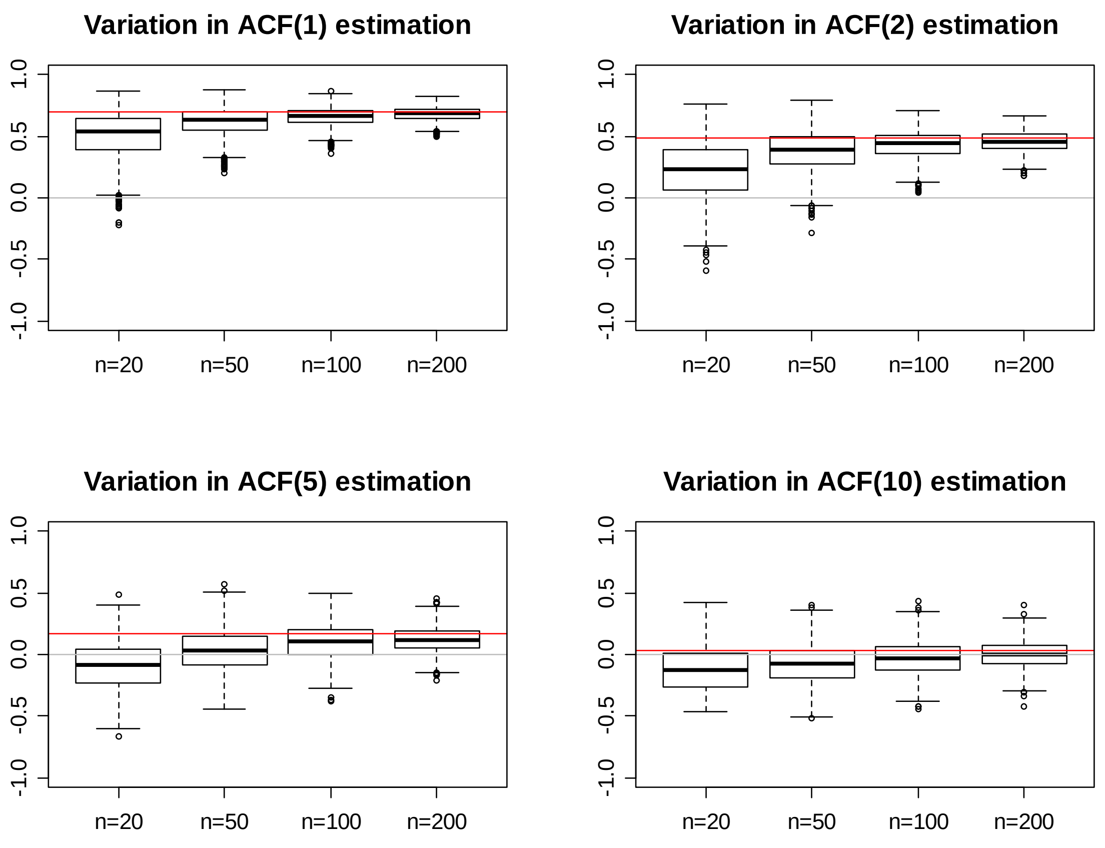
Simulation Study

Last but not least, we will run a small simulation study that visualizes the variability there is in ACF estimation. 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 on the next page.

We observe that for "short" series with less than 100 observations, estimating the ACF is a difficult matter. The $\hat{\rho}(k)$ are strongly biased downwards, 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 decision in the modeling process on the correlogram. However, please be aware of the estimation variability there is.



4.4 Partial Autocorrelation

For the above 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.:

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

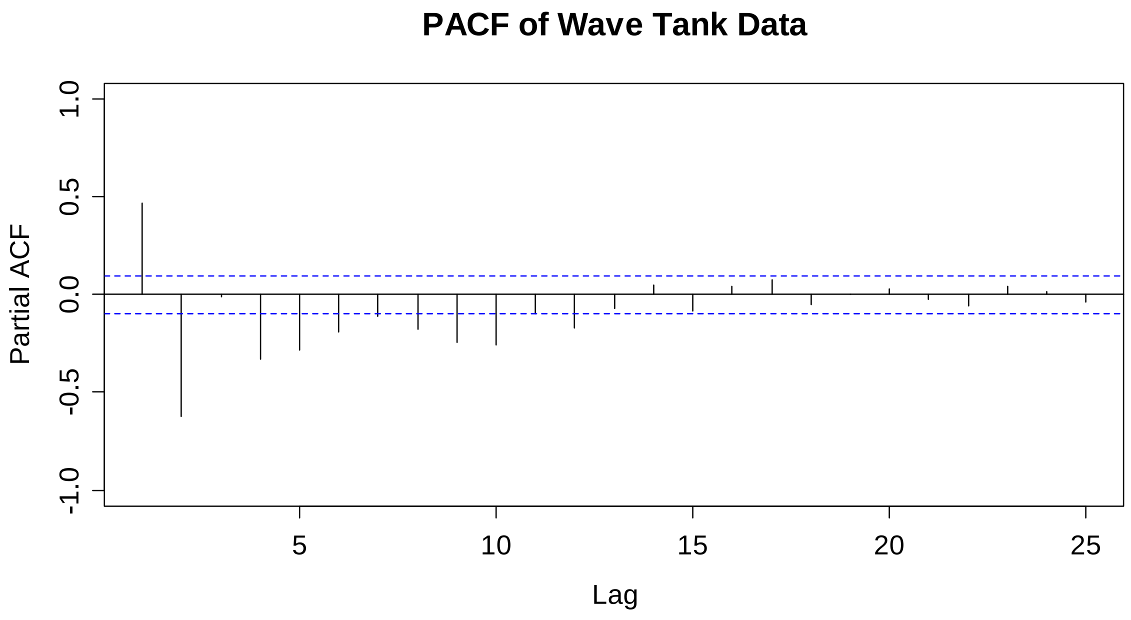
The formula for higher lags k exists, but get complicated rather quickly, so we do without displaying them. However, another absolutely central property of the partial autocorrelations $\pi(p)$ is that the k^{th} coefficent of the AR(p) model, denoted as α_p, is equal to $\pi(p)$. While there is an in 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.

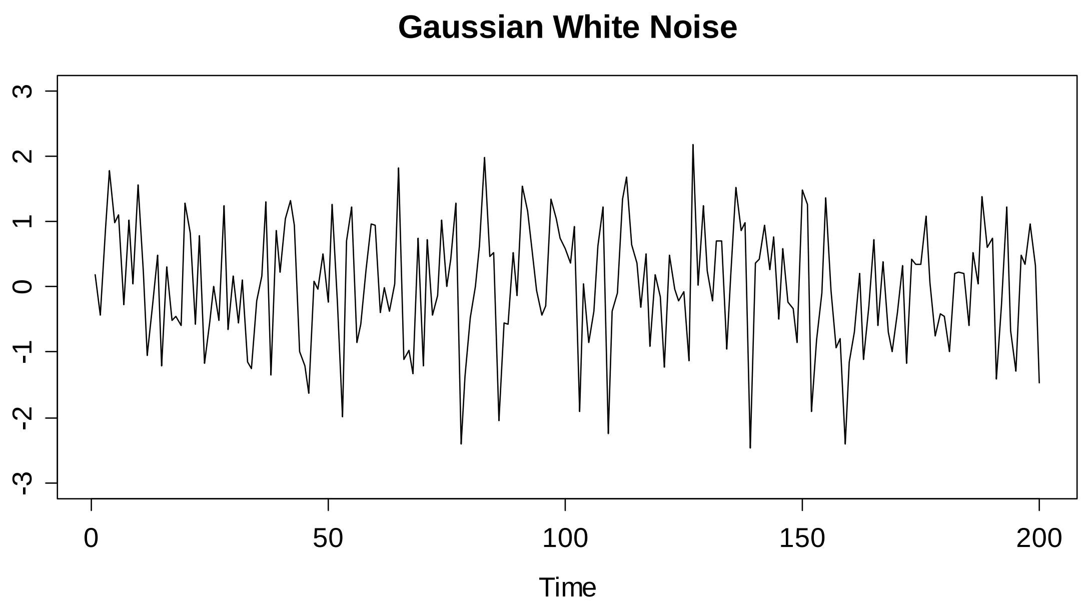
5 Stationary Time Series Models

Rather than simply describing observed time series data, we now aim for fitting 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.

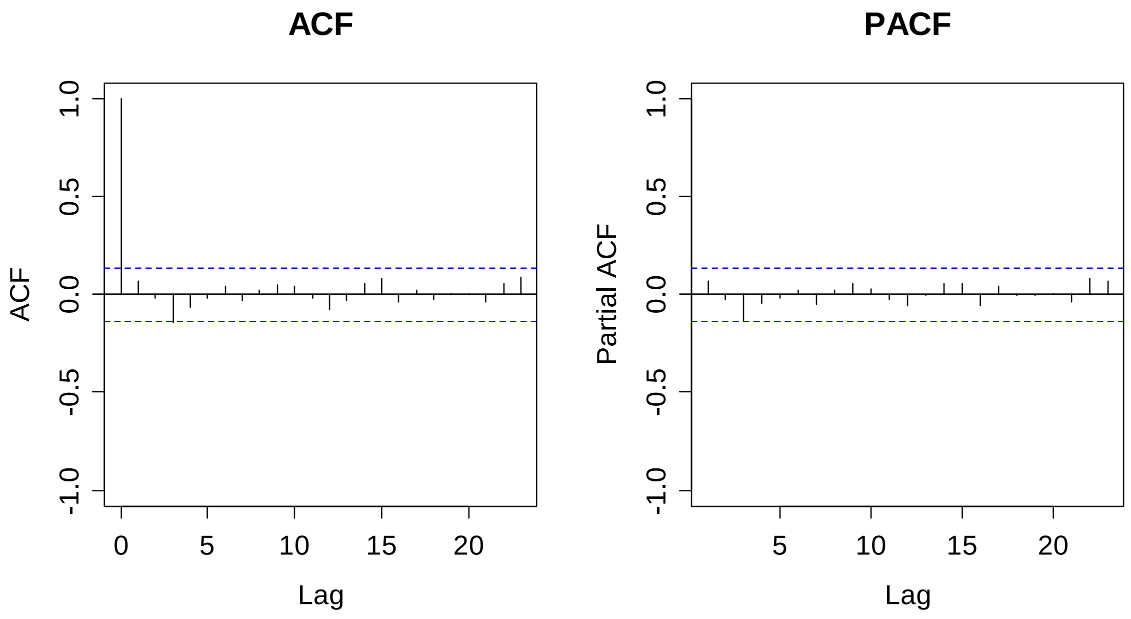
Before we show a realization of a white noise process, we state that the term “white noise” was coined in an article on heat radiation published in Nature in April 1922. There, it was used to refer to series time series that contained all frequencies in equal proportions, analogous to white light. It is possible to show that i.i.d. sequences of random variables do contain all 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 4.3 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 Autoregressive Models

5.3 Moving Average Models

Here, we will discuss moving average models. These can be seen as an extension of the white noise process, i.e. X_t can be written 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 stationary, but not iid. Furthermore, we will see that in many respects, moving average models are complementary to autoregressive models.

5.3.1 Backshift Operator

We start our discussion of moving average models by introducing 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". We can of course apply B repeatedly and so shift back to lag k, in particular:

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

As mentioned above, this will serve us to write time series models in more compact form. We illustrate this for an AR(p) model, where

$X_t = (\alpha_1 B + \alpha_2 B^2 + ... + \alpha_p B^p) X_t + E_t$, or respectively:

$$(1 - \alpha_1 B - \alpha_2 B^2 - ... - \alpha_p B^p) X_t = E_t$$

We can summarize $(1 - \alpha_1 B - \alpha_2 B^2 - ... - \alpha_p B^p)$ by $\Phi(B)$, this is the characteristic polynomial of the respective AR process. Soon, we will exploit the very same mechanism with moving average models.

5.3.2 Model Equation

As we had mentioned above, a moving average model 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 $s < t$. We make use of the backshift operator that was defined above for rewriting the model:

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

Please note that some other textbooks also define this model with negative signs for the β_j. While this is mathematically equivalent, we prefer our notation with the '+' signs, because this is also how things are defined in **R**. Please also note that we can always enhance this model by adding a constant μ that accounts for non-zero expectation of a time series.

Why such MA(q) models? 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 decision, referendums, shortages of key commodities and so on. Such events will not only have an immediate effect, but may also affect the 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 AR processes. This will become clear if we closely study the MA(1) model.

5.3.3 The MA(1) Process

For proofing stationarity and deriving the moments of moving average processes, we first consider the simple model of order 1:

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

where E_t is a white noise process with $E(E_t) = 0$ and $Var(E_t) = \sigma^2$. It is straightforward to show that X_t has mean zero, since it is the sum of two random variables with each mean zero. The variance is also easy to derive:

$$Var(X_t) = (1+\beta_1^2)\sigma_E^2$$

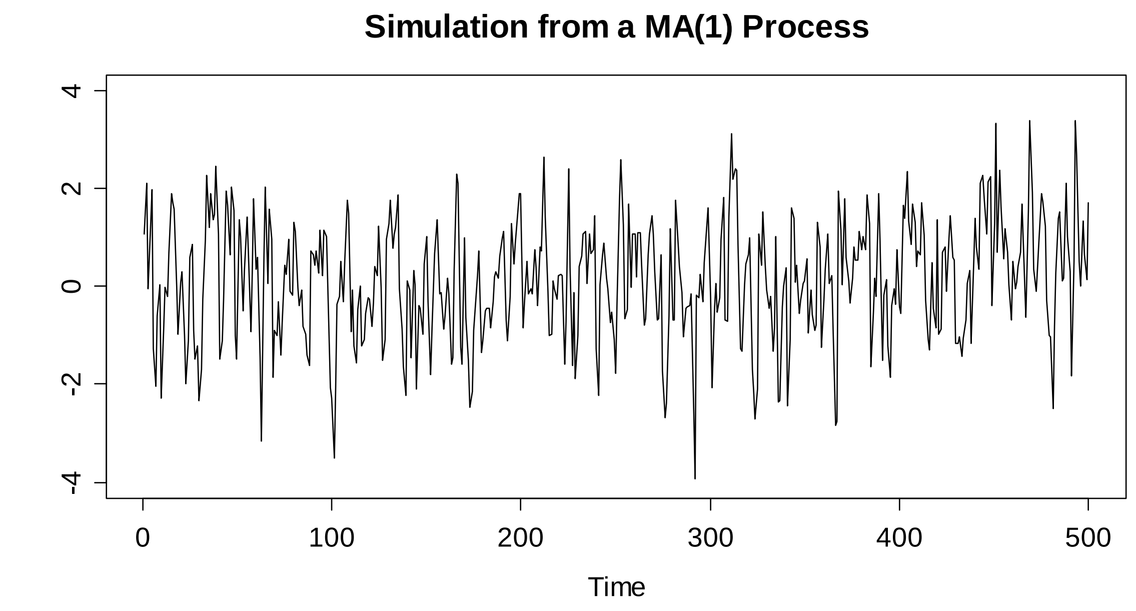
The ACF is special because only the coefficient at lag 1 is different from zero, and there is no further autocorrelation:

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

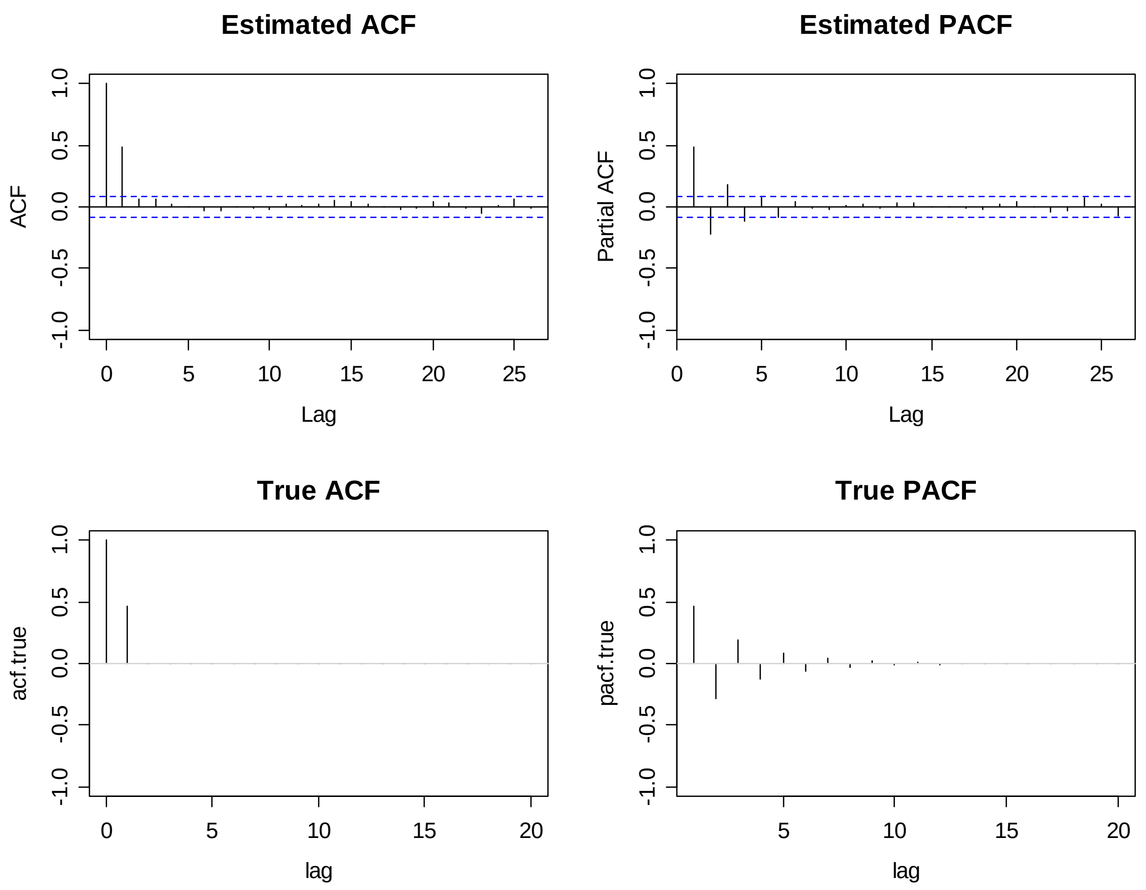
Also, we have $\rho(1) \le 0.5$, no matter what the choice for β_1 is. Thus if in practice we observe a series where the first-order autocorrelation coefficient clearly exceeds this value, we have counterevidence to a MA(1) process.

For illustration, we generate a realization consisting of 500 observations, from such a process with $\beta_1 = 0.7$, and display 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))
> 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 that the estimates are pretty accurate: the ACF has a clear cut-off, whereas the PACF shows some alternating behavior with an exponential decay in absolute value – completely contrary to the stylized facts an AR process shows.

Invertibility

The first autocorrelation coefficient $\rho(1)$ 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, a MA(1) process with coefficient β_1 has exactly the same ACF as the one with $1/\beta_1$. Thus, for example, the two processes $X_t = E_t + 0.5 \cdot E_{t-1}$ and $Y_t = E_t + 2 \cdot E_{t-1}$ have the same dependency structure. This problem of ambiguity leads to the concept of invertibility.

TBC...

5.4 ARMA Models

6 Time Series Regression

6.1 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 $x_t^1,...,x_t^p$. 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:

$$Y_t = \beta_0 + \beta_1 x_t^1 + ... + \beta_p x_t^p + E_t$$

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, the last condition often turns out to be violated.

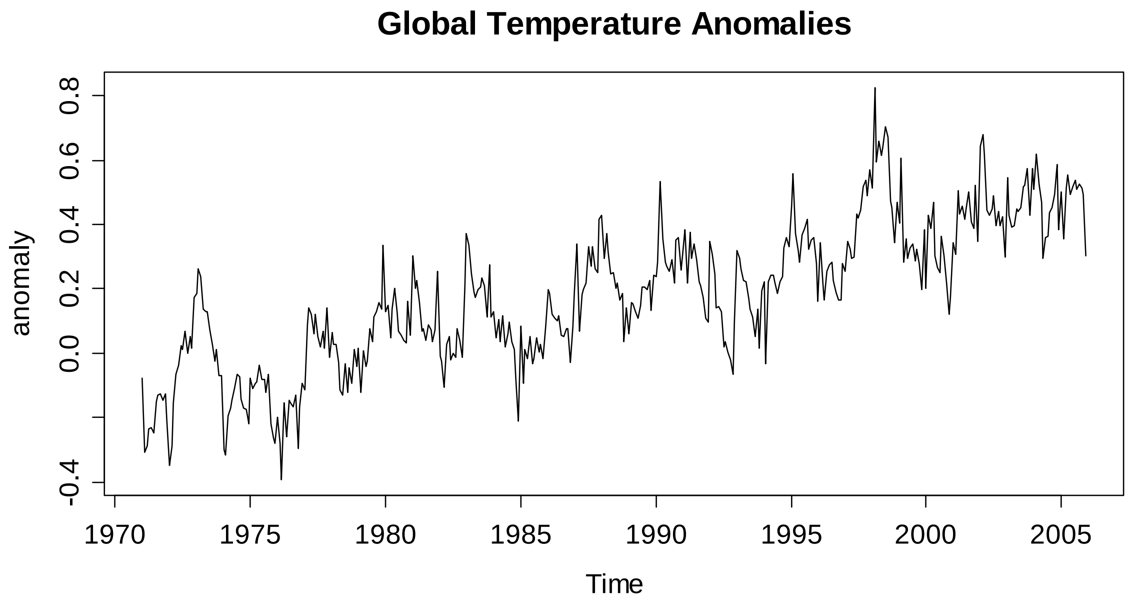
Now, if we are facing a (time series) regression problem with correlated errors, the estimates $\hat{\beta}_j$ will remain being unbiased, but the least squares algorithm is no longer efficient. Or in other words: more precisely working estimators exist. Even more problematic are the standard errors 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 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. For illustrative purposes, we here 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
> my.temp <- window(global, c(1971,1), c(2005,12))
> plot(my.temp, ylab="anomaly")
> title("Global Temperature Anomalies")
```



There is a clear trend which seems to be linear. Despite we have monthly data, there is 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.

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

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.

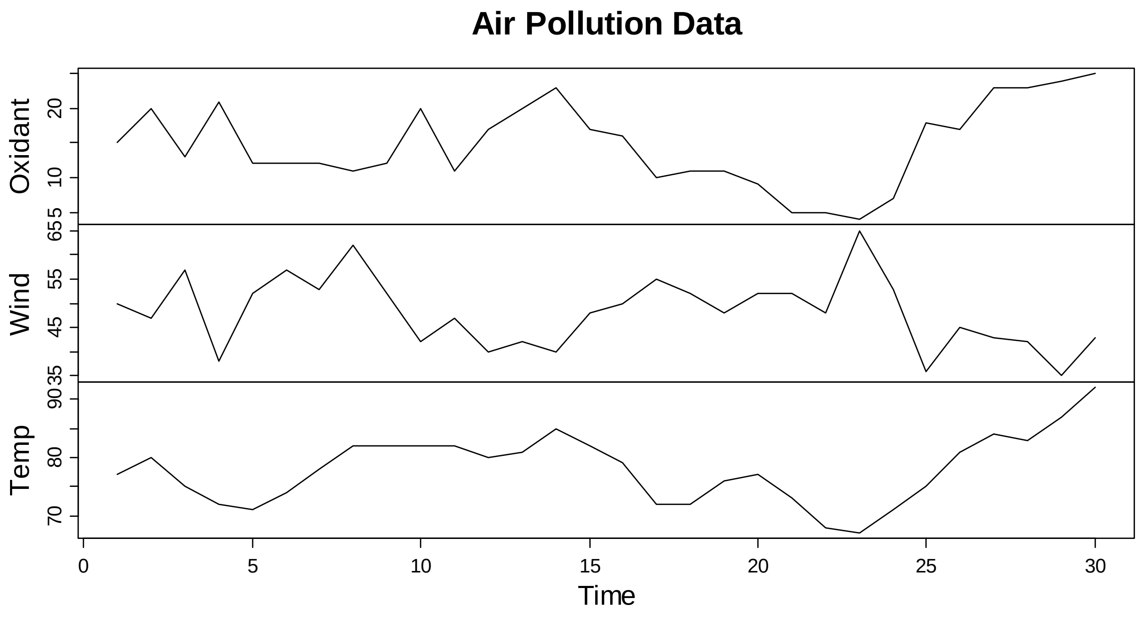
As we have seen previously, the goal with such parametric decomposition models is to obtain a stationary remainder term E_t. But stationary does not necessarily mean white noise, and in practice it often turns out that E_t shows some serial correlation. Thus, if the regression coefficients are obtained 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 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.

```
> ## Importing the data
> tmp <- read.table("pollute.dat", header=TRUE)
> dat <- ts.union(Oxidant=ts(tmp$Oxidant), Wind=ts(tmp$Wind),
                  Temp=ts(tmp$Temp))
> ## Visualizing the data
> plot(dat, main="Air Pollution Data")
```

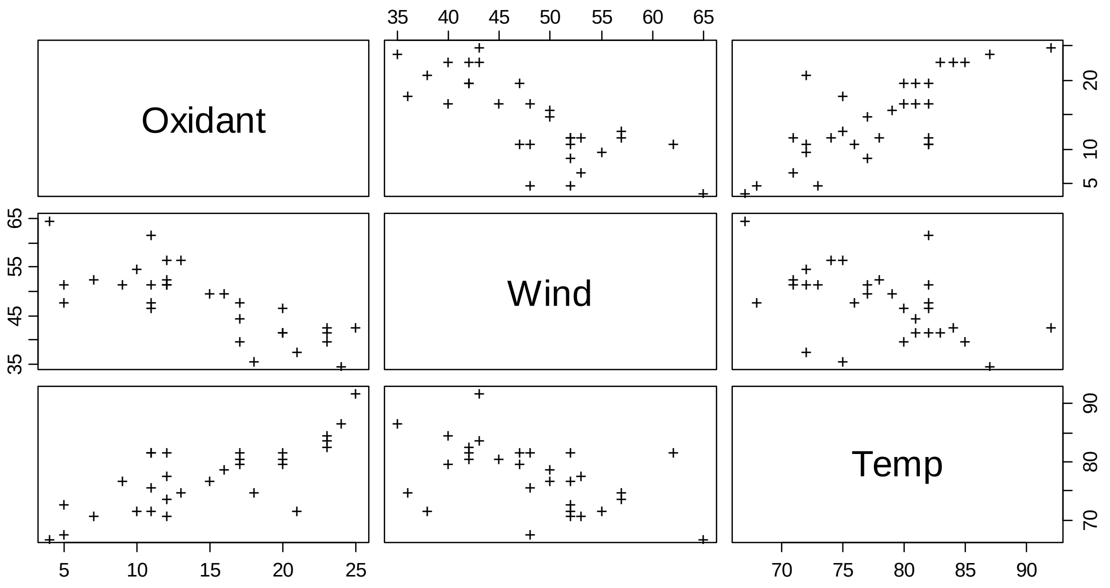


There is no counterevidence to stationarity for all three series. What could be the goal here? Well, we aim for enhancing the understanding of how the pollution depends on the meteorology, i.e. what influence wind and temperature have on the oxidant values. We can naturally formulate the relation with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

$$Y_t = \beta_0 + \beta_1 x_t^1 + \beta_2 x_t^2 + E_t \,.$$

In this model, the response Y_t is the oxidant, and as predictors we have x_t^1, wind, and x_t^2, 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

$$Y_t = \beta_0 + \beta_1 x_t^1 + ... + \beta_p x_t^p + E_t$$

we assume that the influence of the series $x_t^1, \ldots, x_t^q$ 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 later in section **Fehler! Verweisquelle konnte nicht gefunden werden.**.

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_t^j. Additionally, the error E_t must be independent of the explanatory variables, but it may exhibit serial correlation.

6.2 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(my.temp)
> num.time <- as.numeric(time(my.temp))
> 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)
```

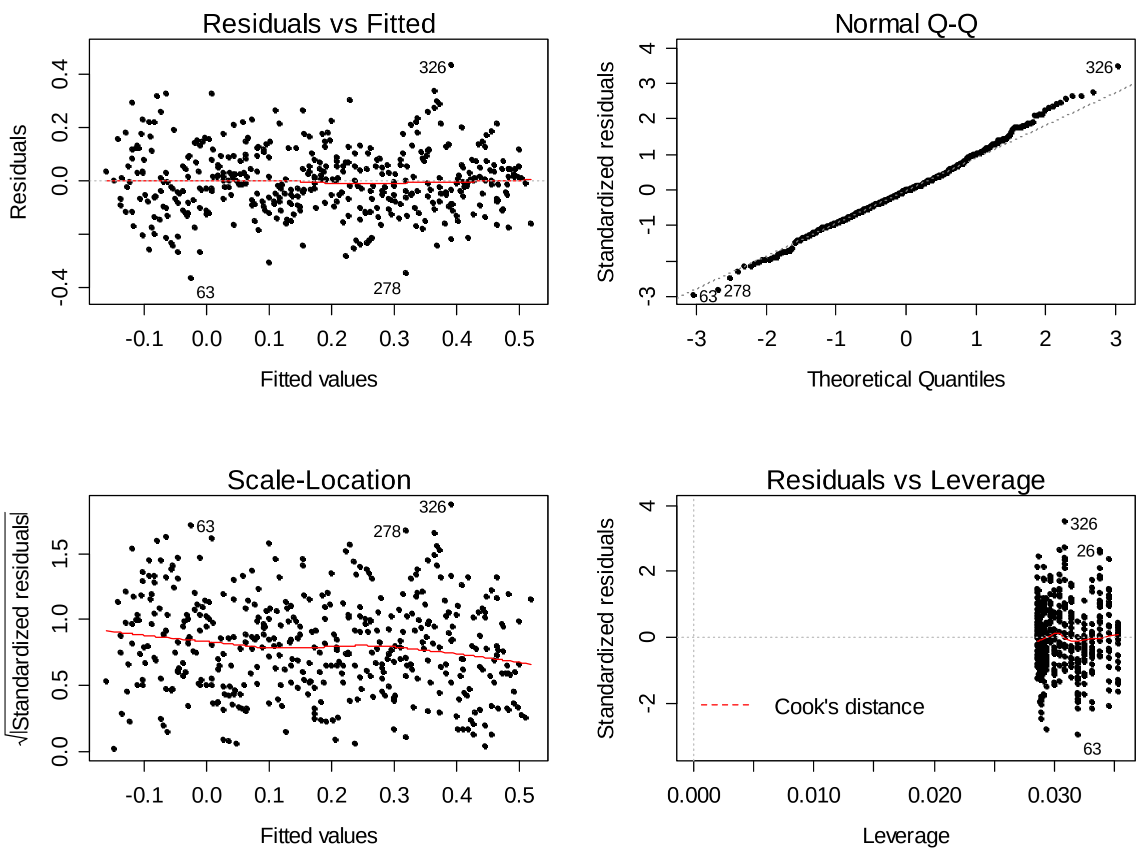
The regression model is the estimated with **R**'s function `lm()`. The summary function returns estimates, standard errors plus the results from some hypothesis tests. It is important to notice that all of these results are in question should the errors turn out to be correlated.

```
> 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
```

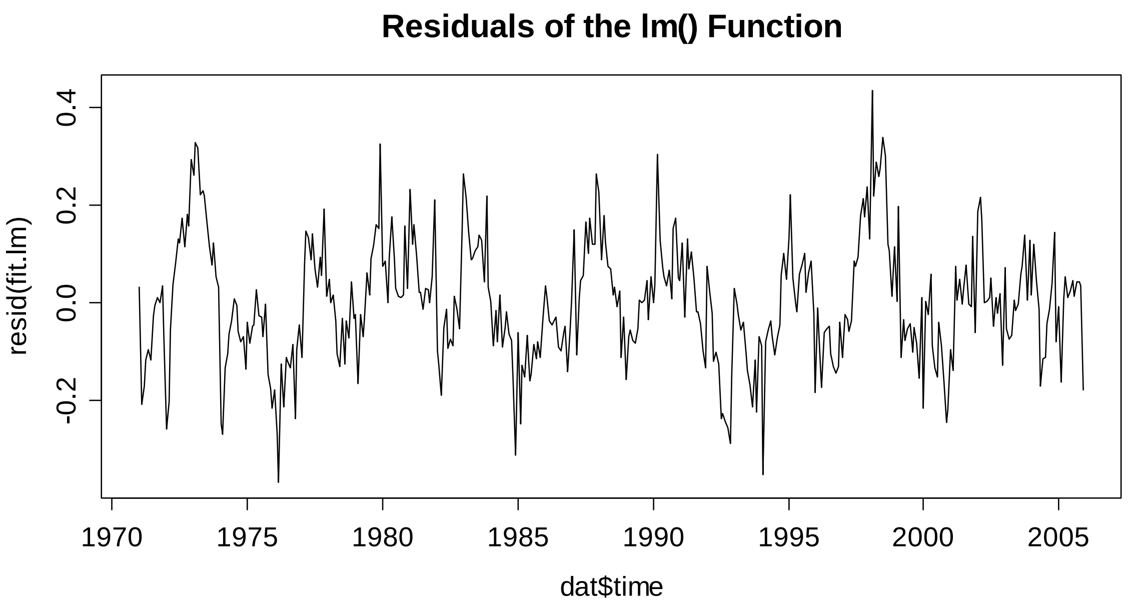
As the next step, we need to perform some residual diagnostics. The `plot()` function, applied to a regression fit, serves as a check for zero expectation, constant variation and normality of the errors, and can give hints on potentially problematic leverage points.

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



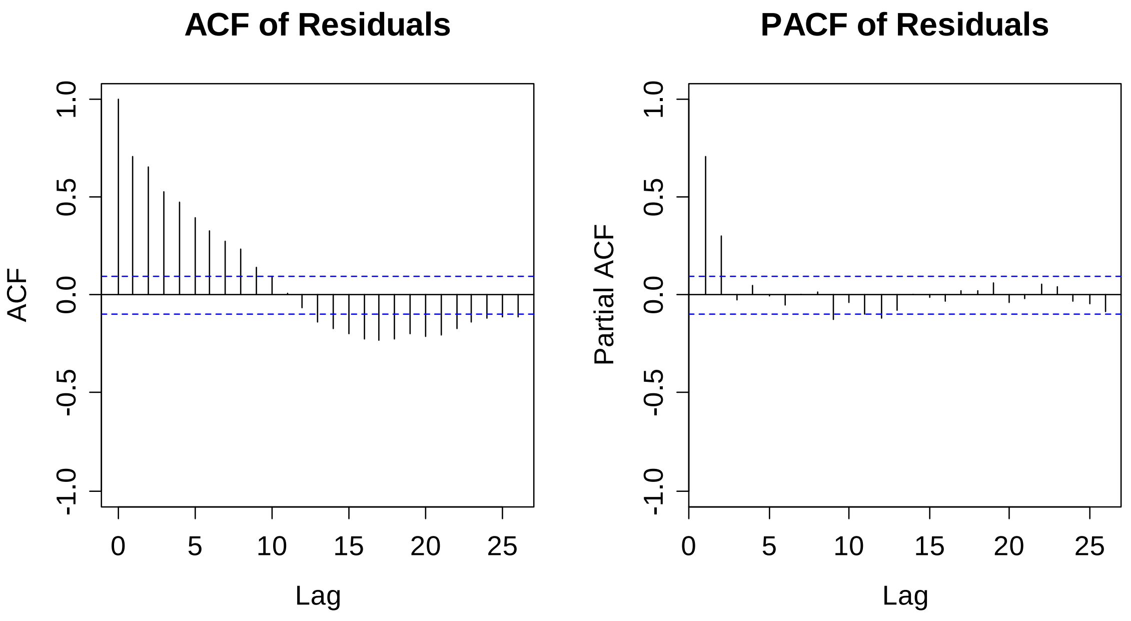
Except for some very slightly long tailed errors, which do not require any action, the residual plots look fine. What has not yet been verified is whether there is any serial correlation among the residuals. If we wish to see a time series plot, the following commands are useful:

```
> plot(dat$time, 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:

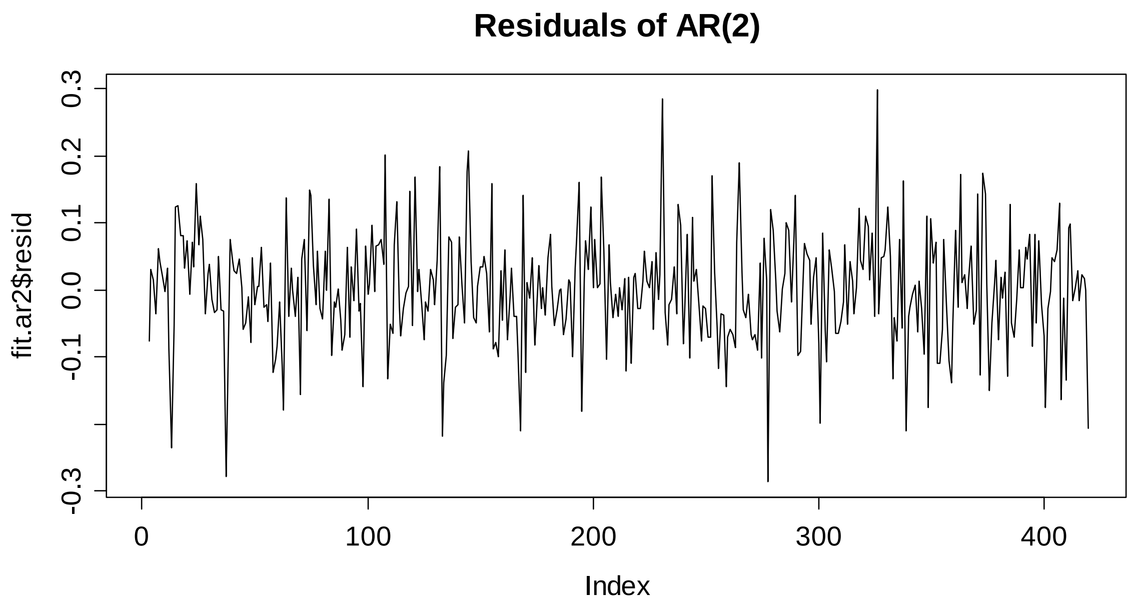
```
> par(mfrow=c(1,2))
> acf(resid(fit.lm), main="ACF of Residuals")
> pacf(resid(fit.lm), main="PACF of Residuals")
```

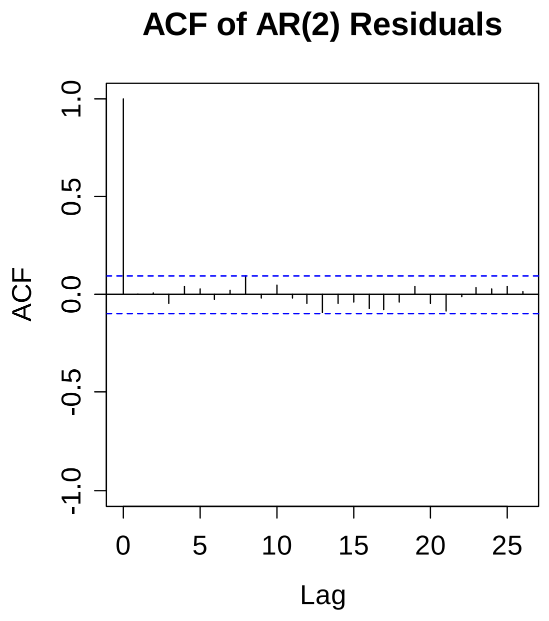


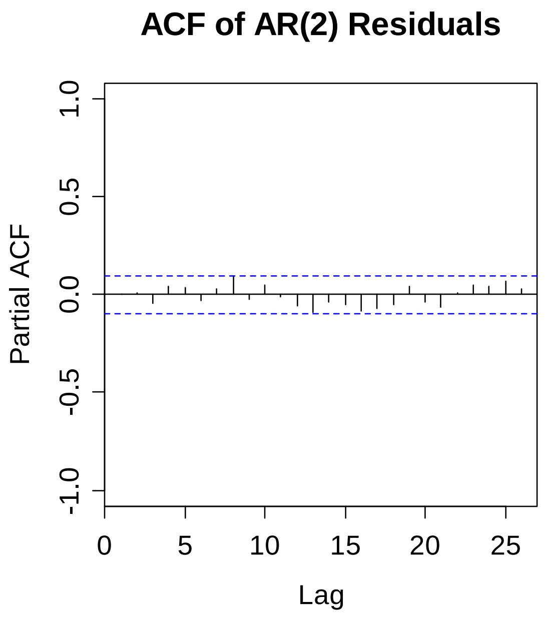
The ACF shows a rather slow exponential decay, whereas the PACF shows a clear cut-off at lag 2. With these stylized facts, it might well be that an AR(2) model is a good description for the dependency among the residuals. We verify this:

```
> 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.

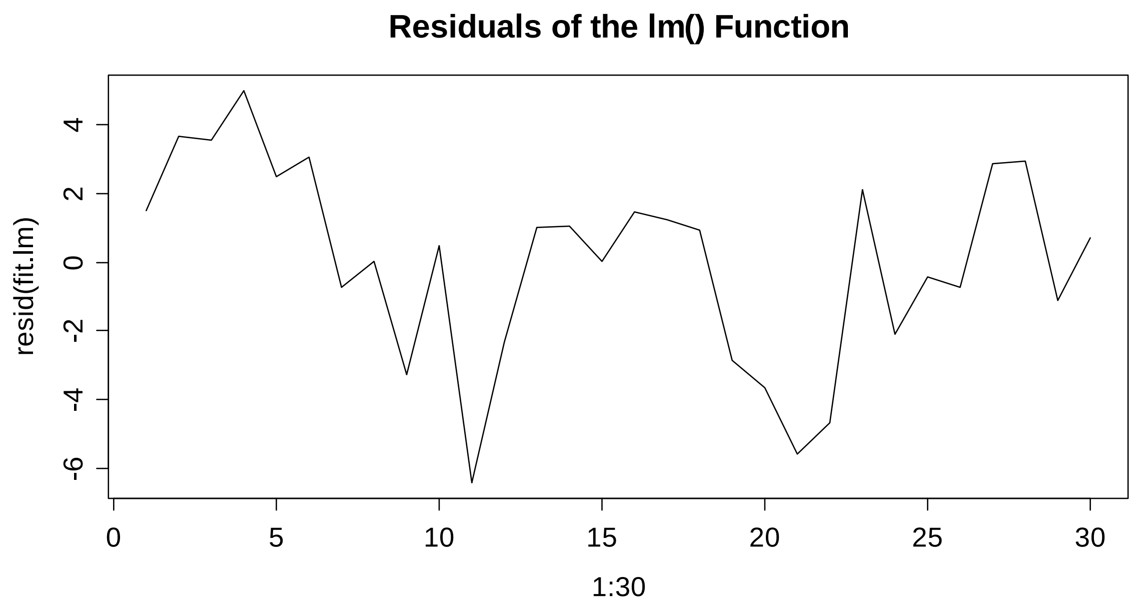
Example 2: Air Pollution

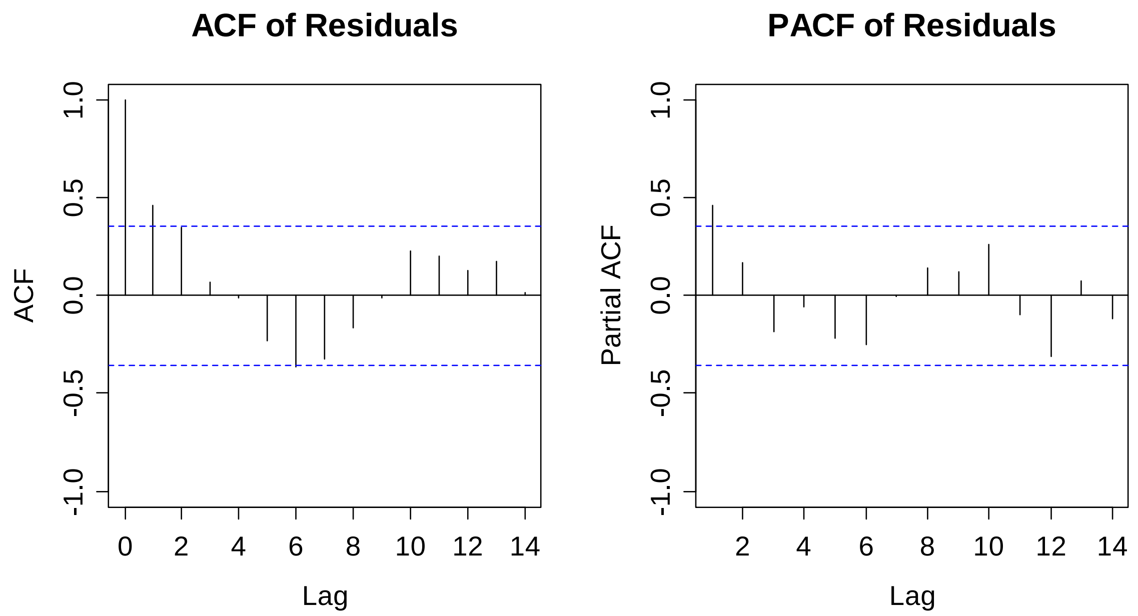
Now, we are dealing with the air pollution data. Again, we begin our regression analysis using the standard assumption of uncorrelated errors. Thus, we start out by applying the `lm()` function and printing the `summary()`.

```
> 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.

6.2.1 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.

6.3 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

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 $\beta_1^{MA(q)}, ..., \beta_q^{MA(q)}$. This is exactly what we have done for our two examples: we fitted OLS regressions, and identified an AR(2), respectively an AR(1) dependency among the residuals.

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.

While the Cochrane-Orcutt procedure has its historical importance and is nice for illustration, it lacks of a direct **R** implementation, and, as an iterative procedure, also of mathematical closeness and quality. Thus, we do without performing Cochrane-Orcutt with 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.

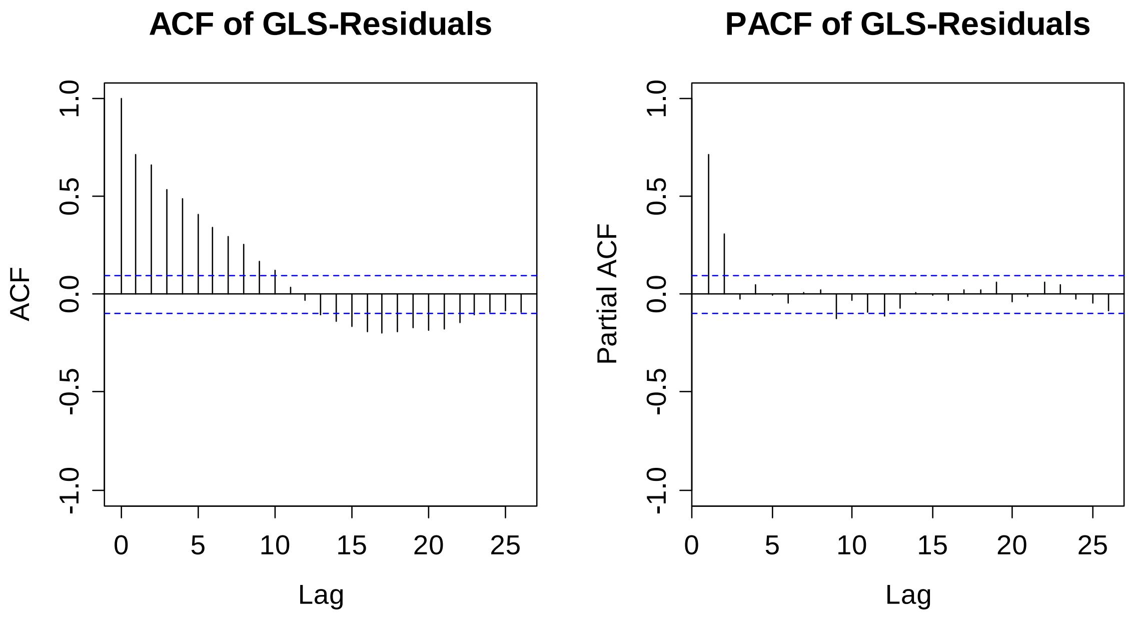
Example 1: Global Temperature

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 a ARMA(p,q) process 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.

Example 2: Air Pollution

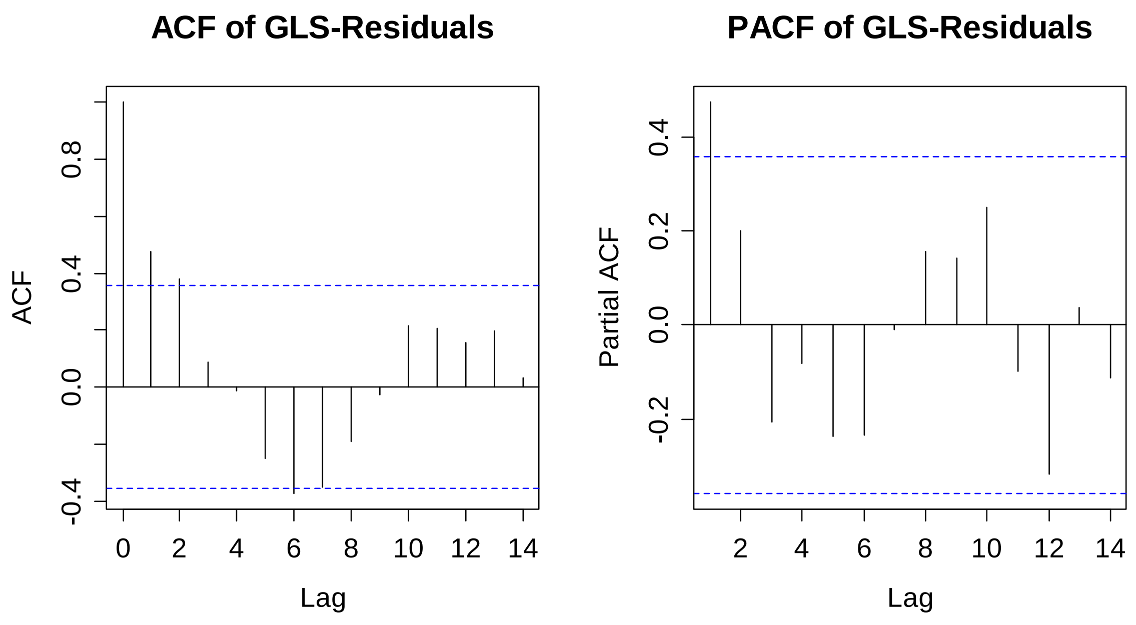
For finishing the air pollution example, we also perform a GLS fit here. We identified an AR(1) as the correct dependency structure for the errors. Thus, we define it accordingly:

```
> 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
```

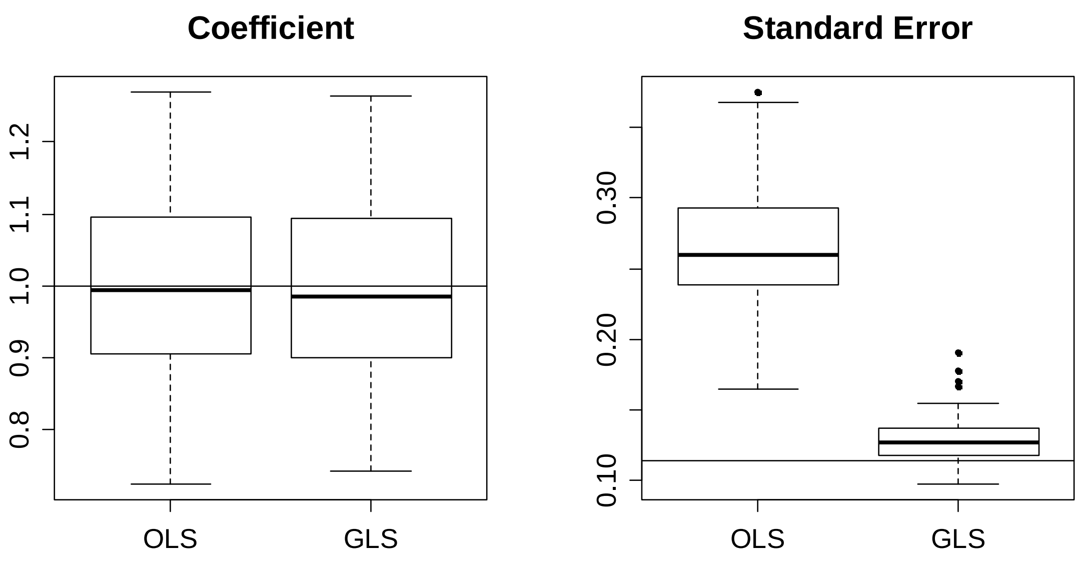
Here the differences among point estimates and confidence intervals are not very pronounced. This has to do with the fact that the correlation among the errors is weaker than in the global temperature example. But we emphasize again that the GLS results are the one to be relied on and the magnitude of 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 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 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.

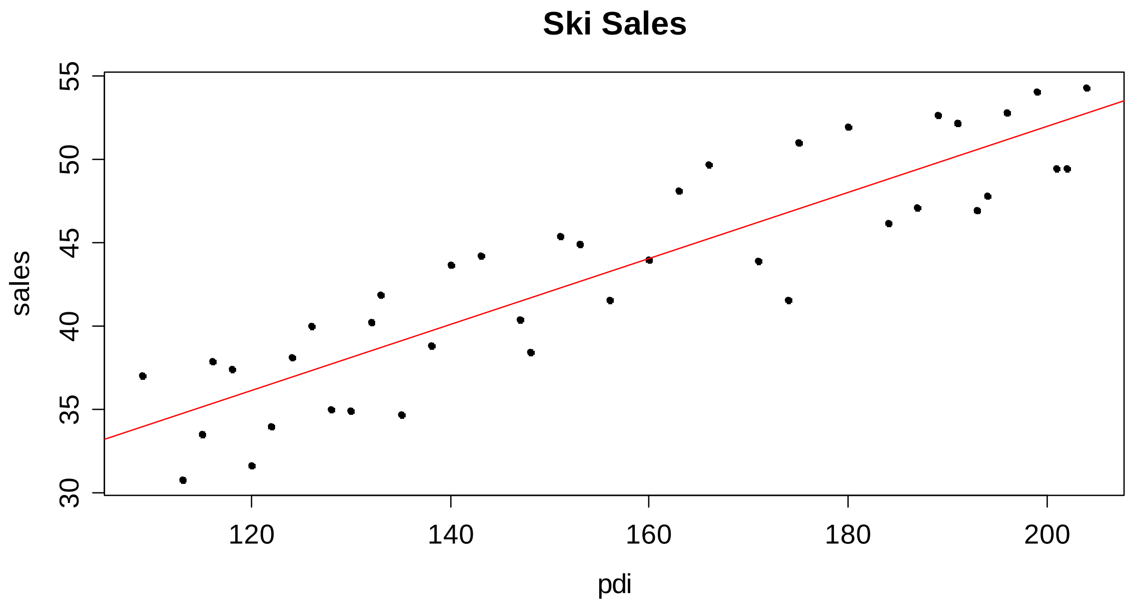


The simulation outcome is displayed by the boxplots in the figure above. While the point estimator for β_1 in the left panel is unbiased for both OLS and GLS, we observe that the standard error for $\hat{\beta}_1$ is very poor when the error correlation is not accounted for. We emphasize again that OLS regression with time series will inevitably lead to spuriously significant predictors and thus, false conclusions. With the next subsection, we conclude the chapter about time series regression by showing how correlated errors do appear, and what practice has to offer for deeper understanding of the problem.

6.4 Missing Predictor Variables

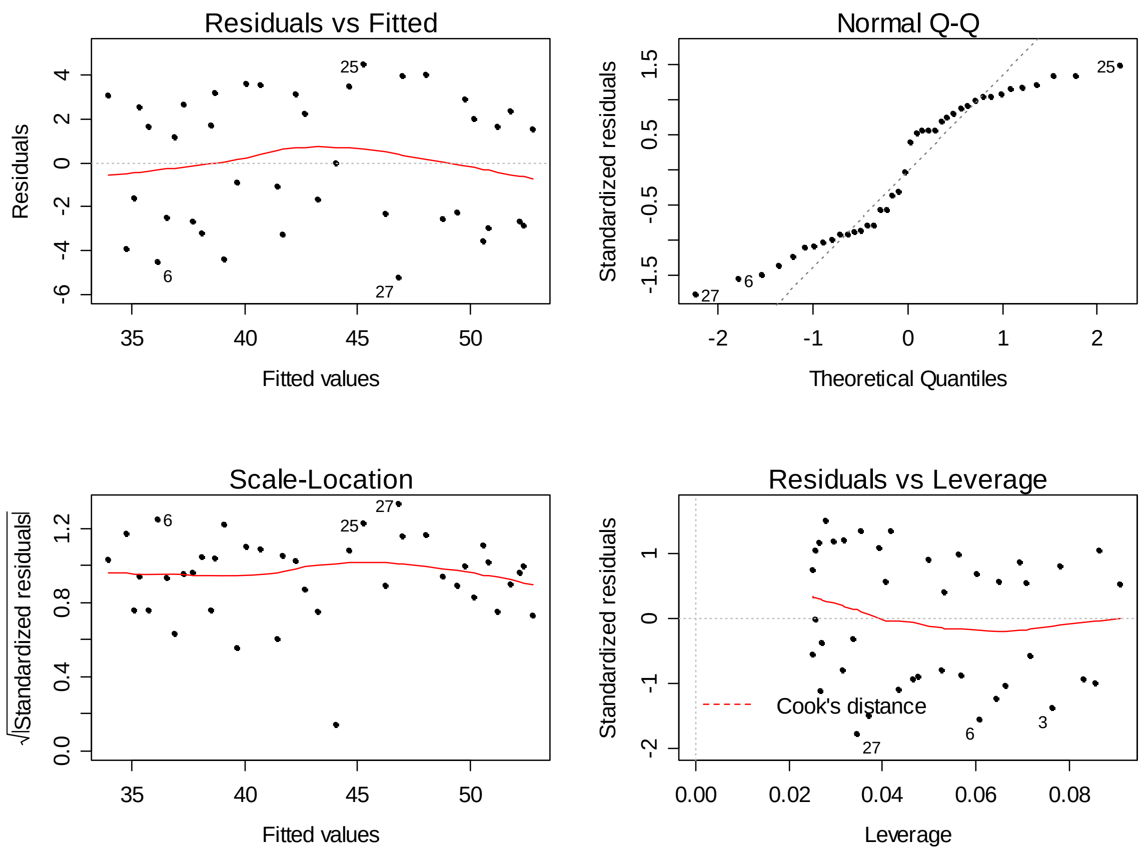
The presence correlated errors is often due to missing predictors. For illustration, we consider a straightforward example of a ski selling company in the US. The quarterly sales Y_t are regressed on the personal disposable income (PDI) which is the one and only predictor x_t. We display the two time series in a scatterplot and enhance it with the OLS regression line.

```
> ## Loading the data
> ski        <- read.table("ski.dat",header=TRUE)
> names(ski) <- c("time", "sales", "pdi", "season")
>
> ## Scatterplot
> par(mfrow=c(1,1))
> plot(sales ~ pdi, data=ski, pch=20, main="Ski Sales")
>
> ## LS modeling and plotting the fit
> fit <- lm(sales ~ pdi, data=ski)
> abline(fit, col="red")
```



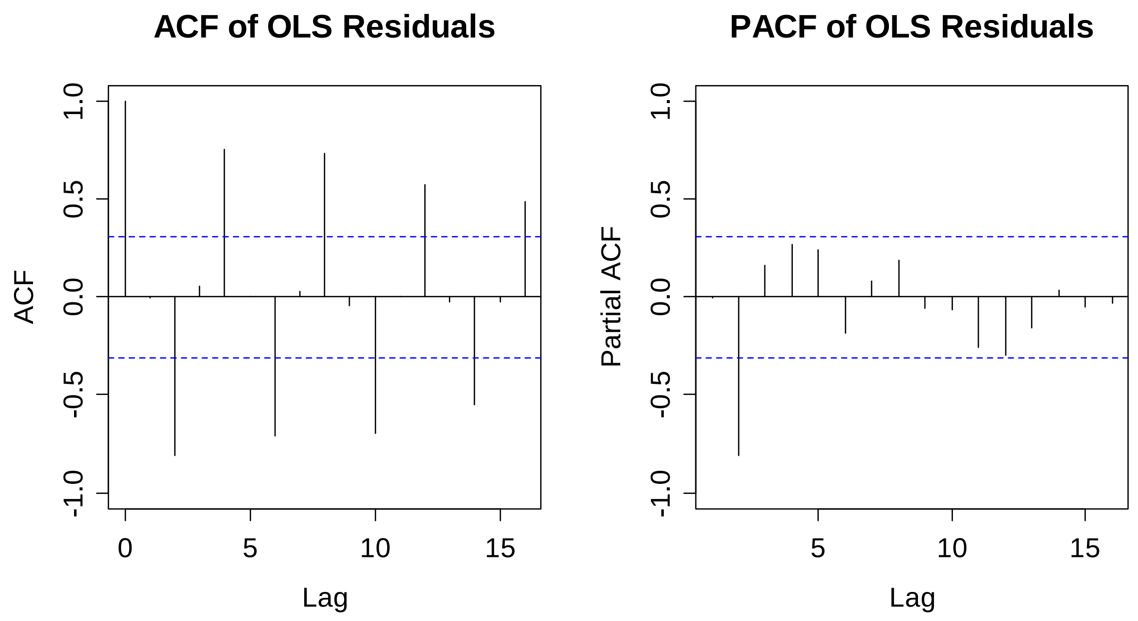
The coefficient of determination is rather large, i.e. $R^2 = 0.801$ and the linear fit seems adequate, i.e. a straight line seems to correctly describe the systematic relation between sales and PDI. However, the model diagnostic plots (see the next page) show some rather special behavior, i.e. there 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)
```

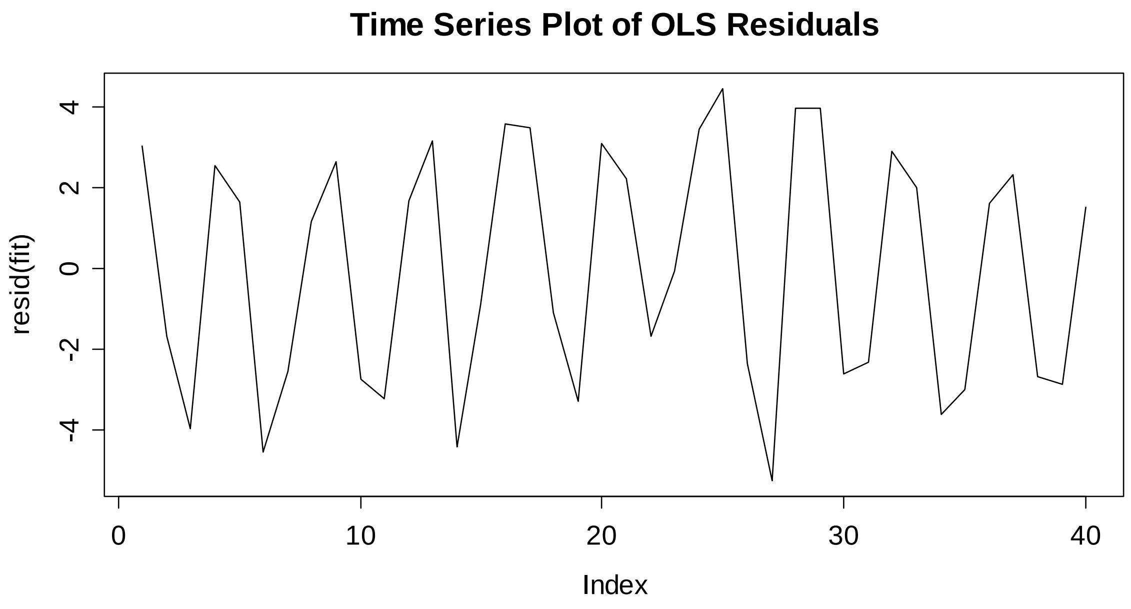


As the next step, we analyze the correlation of the residuals and perform a Durbin-Watson test. The result is as follows:

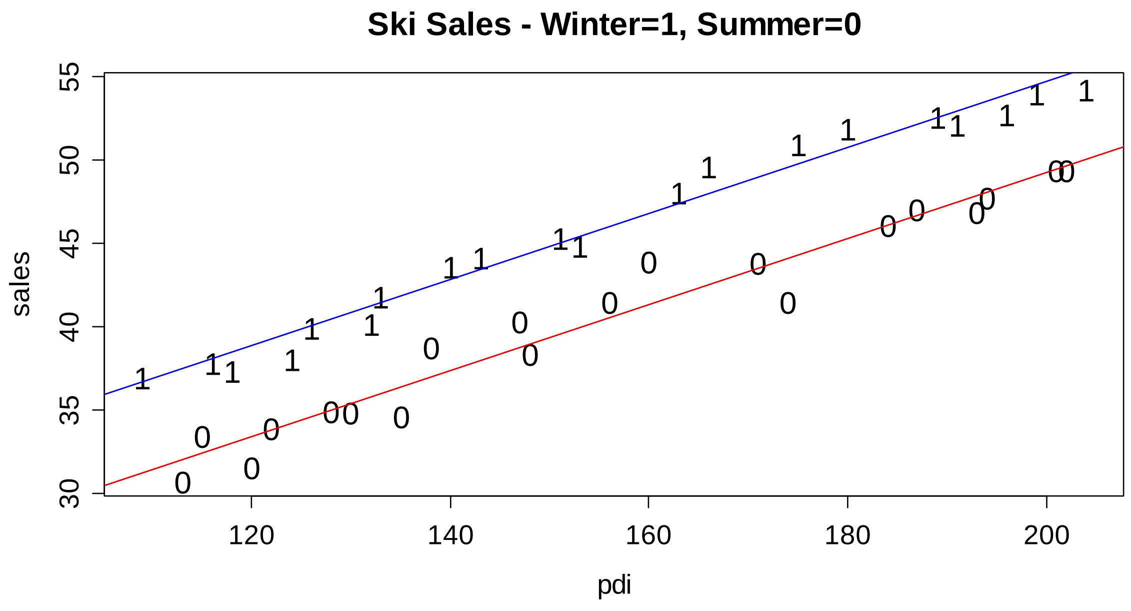
```
> dwtest(fit)
data:  fit
DW = 1.9684, p-value = 0.3933
alt. hypothesis: true autocorrelation is greater than 0
```



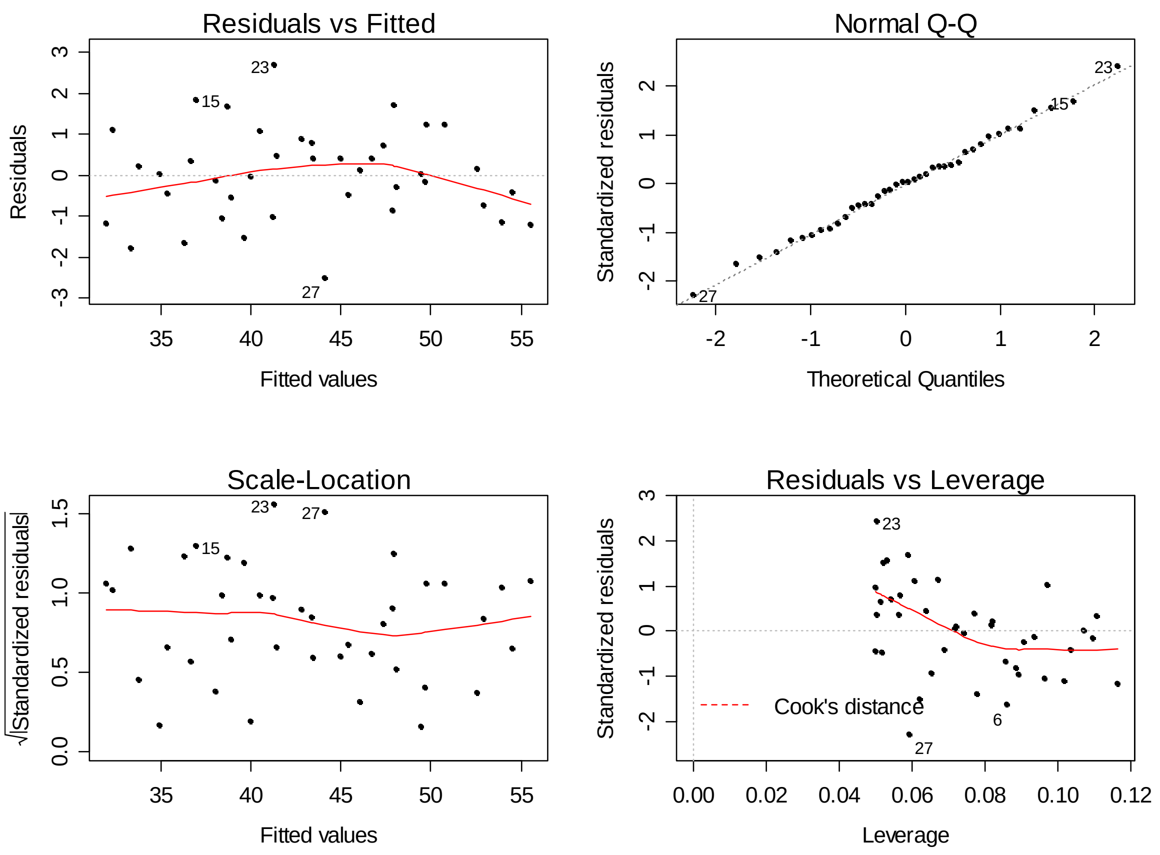
While the Durbin-Watson test does not reject the null hypothesis, the residuals seem very strongly correlated. The ACF exhibits some decay that may still qualify as exponential, and the PACF has a clear cut-off at lag 2. Thus, an AR(2) model could be appropriate. And because it is an AR(2) where α_1 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:



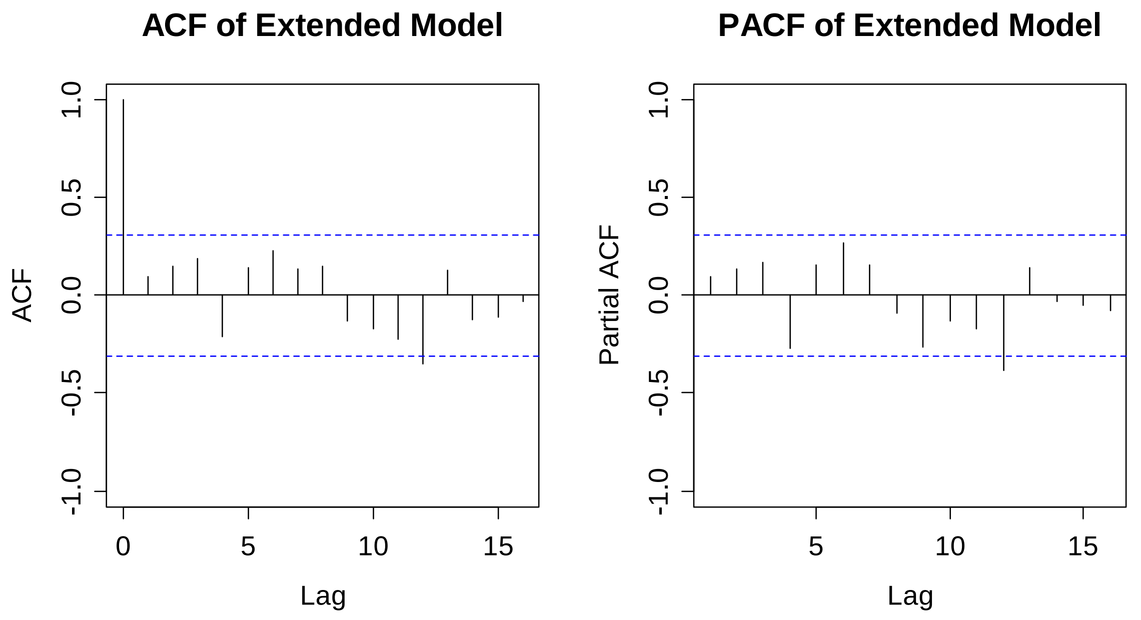
While we could now account for the error correlation with a GLS, it is always better to identify the reason behind the dependence. I admit this is suggestive here, but as mentioned in the introduction of this example, these are quarterly data and we might have forgotten to include the seasonality. It 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 parallel regression lines for the winter and summer term. 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.

7 Non-Stationary Models

As we have discovered previously, many time series are non-stationary due to deterministic trends and/or seasonal effects. While we have learned to remove these and then explain the remainder with some time series models, there are other processes that 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 may also be non-stationary because the variance is serially correlated, i.e. they are conditionally heteroskedastic. Such series, often from financial or economic background, usually 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 variance. These are the famous ARCH models, which we will discuss along with their generalized variant, the GARCH class.

7.1 ARIMA Models

ARIMA models are aimed at describing series which exhibit a deterministic trend that can be removed by differencing; 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 dth order difference of X_t is an ARMA(p,q) process. If we introduce

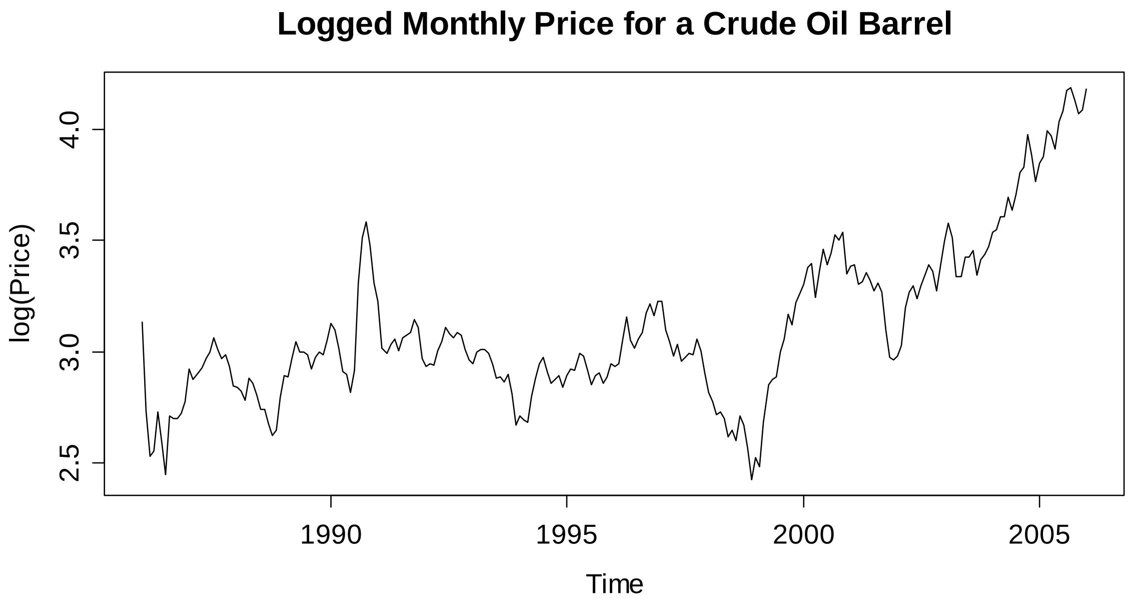
$$Y_t = (1-B)^d X_t \,,$$

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 AR, and $\Phi(\cdot)$ that stands for the MA part.

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

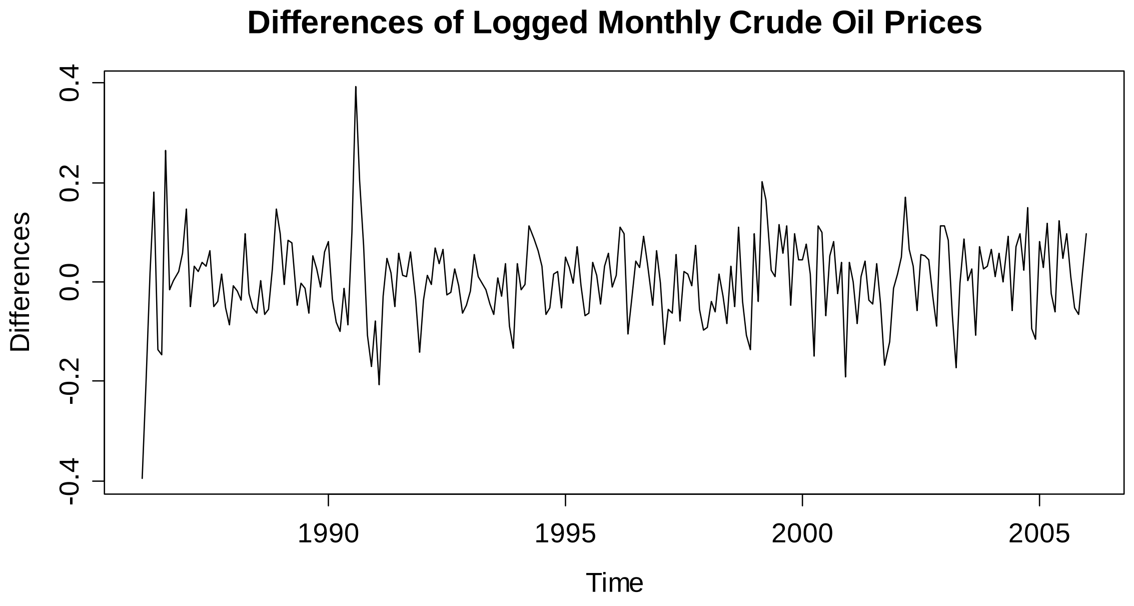
Such series do appear in practice, as our example of the monthly prices for a barrel of crude oil (in US$) from January 1986 to January 2006 shows. To stabilize the variance, we decide to log-transform the data, and model these.

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



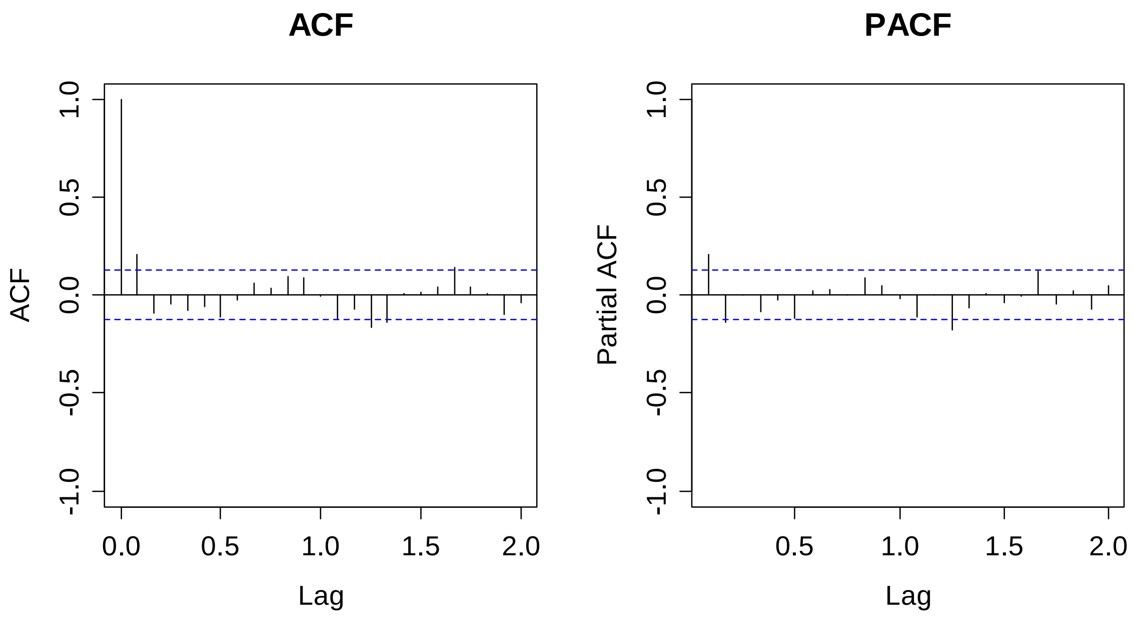
The series does not exhibit any apparent seasonality, but there is a clear trend, so that it is non-stationary. We could assume a piecewise linear trend and try first-order (i.e. $d = 1$) differencing, and then check whether the result is stationary.

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



The trend was successfully removed by taking differences. When we investigate ACF and PACF, we conclude that the differences are not iid, but dependent. There is a drop-off in the ACF at lag 1, and in the PACF at either lag 1 or 2, and thus for the logged differences, an ARMA(1,1) or an ARMA(1,2) could be appropriate. This means an ARIMA(1,1,1) or ARIMA(1,1,2) for the logged oil prices.

```
> 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(1,1,2))

Call:
arima(x = lop, order = c(1, 1, 2))

Coefficients:
         ar1      ma1      ma2
      0.8429  -0.5730  -0.3104
s.e.  0.1548   0.1594   0.0675

sigma^2 = 0.006598:  log likelihood = 261.88,  aic = -515.75
```

Or, we can use the differenced series `dlop` as input and fit an ARMA(1,2). However, we need to tell **R** to not include an intercept – this is not appropriate when a piecewise linear trend was removed by taking differences. The command is:

```
> arima(dlop, order=c(1,0,2), include.mean=FALSE)
```

The output from this is exactly the same as above. The next step is to perform residual analysis – if the model is appropriate, they must look like white noise. This is (data not shown here) more or less the case. 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:

$$\begin{aligned}
Y_t &= 0.84 \cdot Y_{t-1} + E_t - 0.57 \cdot E_{t-1} - 0.31 \cdot E_{t-2} \\
X_t - X_{t-1} &= 0.84 \cdot (X_{t-1} - X_{t-2}) + E_t - 0.57 \cdot E_{t-1} - 0.31 \cdot E_{t-2} \\
X_t &= 1.84 \cdot X_{t-1} - 0.84 \cdot X_{t-2} + E_t - 0.57 \cdot E_{t-1} - 0.31 \cdot E_{t-2}
\end{aligned}$$

Thus, the ARIMA(1,1,2) can be rewritten as a non-stationary ARMA(2,2). The non-stationarity is due to the roots of the AR characteristic polynomial, which are within the unit circle. Finally, we give some recipe for fitting ARIMA models:

1) Choose the appropriate order of differencing, usually $d=1$ or $d=2$, 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.

7.2 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

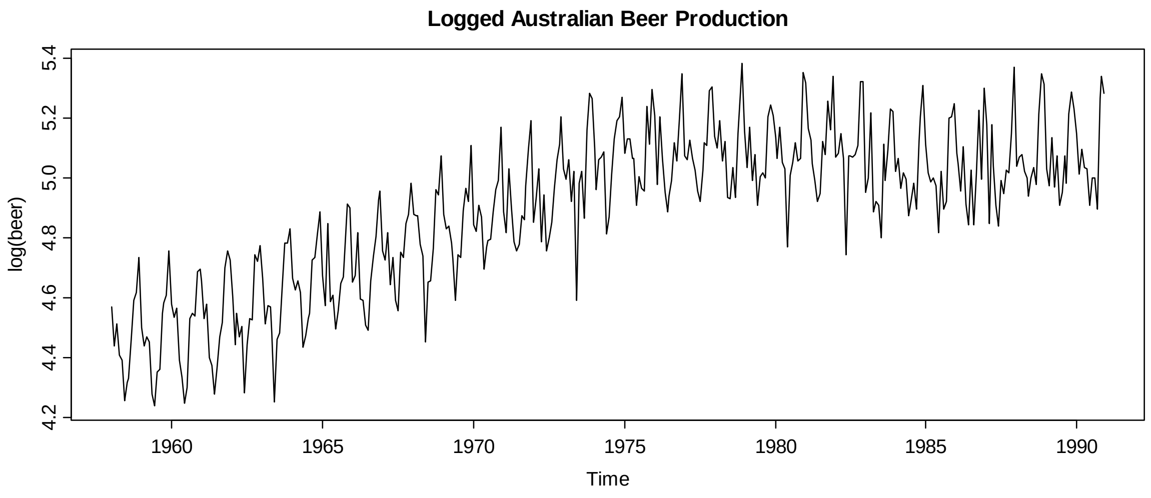
$$Y_t = X_t - X_{t-12} = (1-B^{12})X_t$$

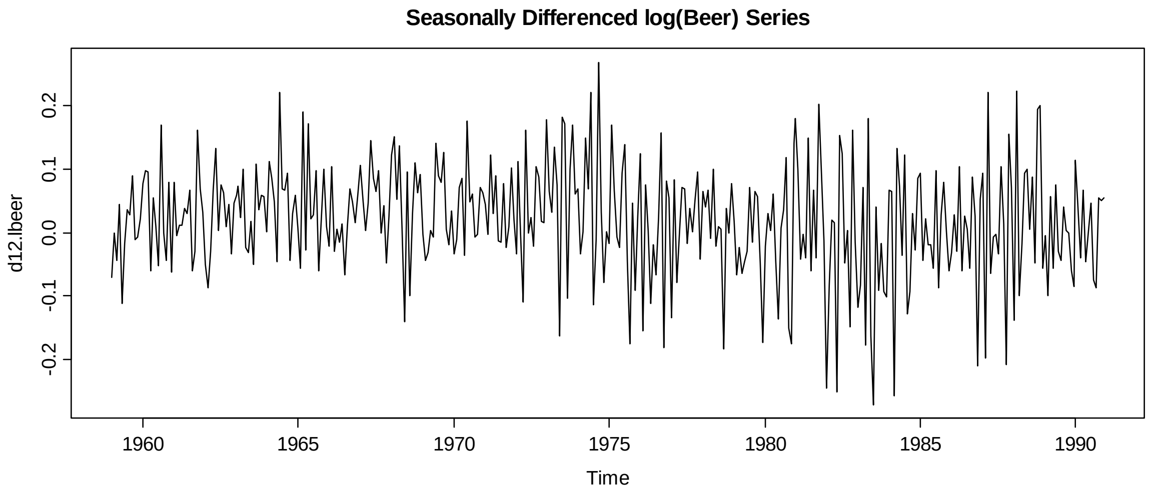
has the seasonality removed. However, it is quite often the case that the result has not yet constant mean, and thus, some further differencing at lag 1 is required to achieve stationarity:

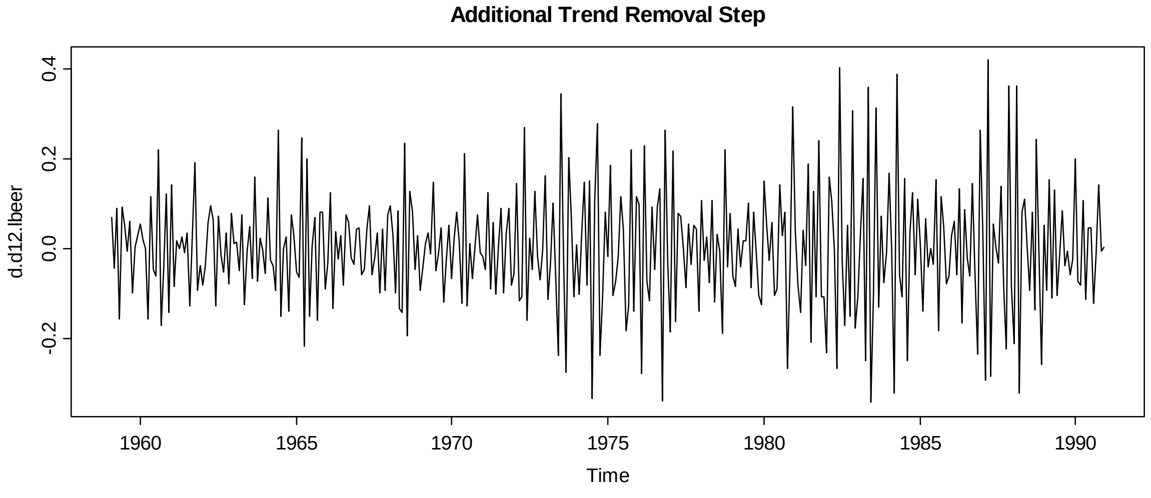
$$Z_t = Y_t - Y_{t-1} = (1-B)Y_t = (1-B)(1-B^{12})X_t = X_t - X_{t-1} - X_{t-12} + X_{t-13}$$

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. Again, a log-transformation to stabilize the variance is indicated. On the next page, we display the original series X_t, the seasonally differenced series Y_t and finally the seasonal-trend differenced series Z_t.

```
> www        <- "http://www.massey.ac.nz/~pscowper/ts/cbe.dat"
> dat        <- read.table(www, header=TRUE)
> 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))
```

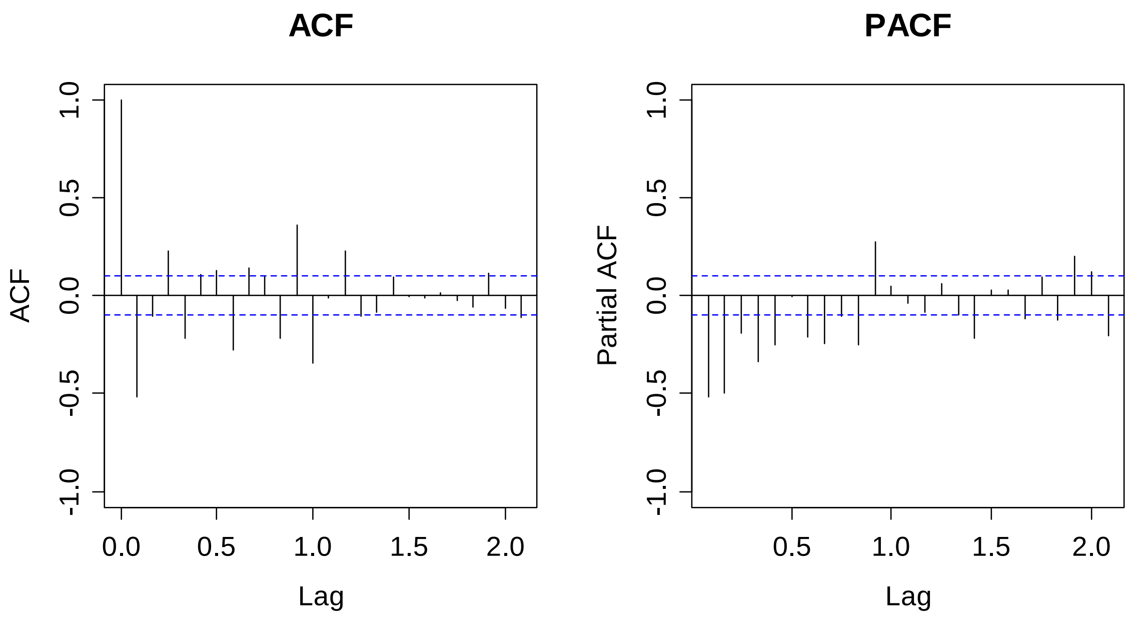






While the two series X_t and Y_t are non-stationary, the last one, Z_t may be, although it is a bit debatable whether the assumption of constant variation is violated or not. We proceed by analyzing ACF and PACF of series Z_t.

```
> 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 coefficient that 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 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:

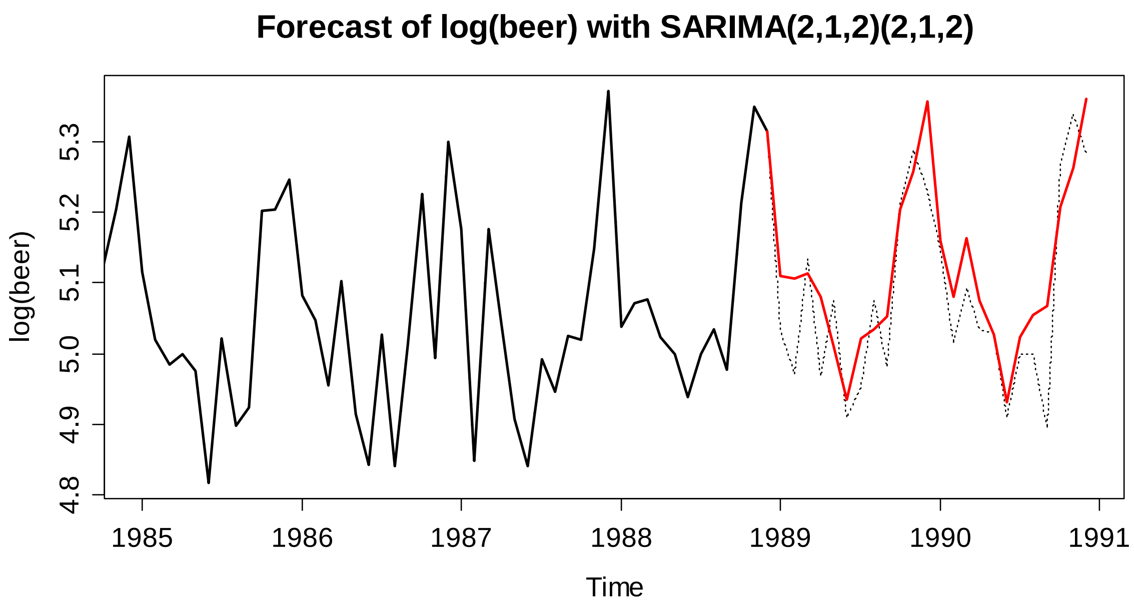
$$\Phi(B)\Phi_S(B^s)Z_t = \Theta(B)\Theta_S(B^s)E_t,$$

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 $p, q, P, Q \le 2$ and doing an AIC-based grid search.

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.

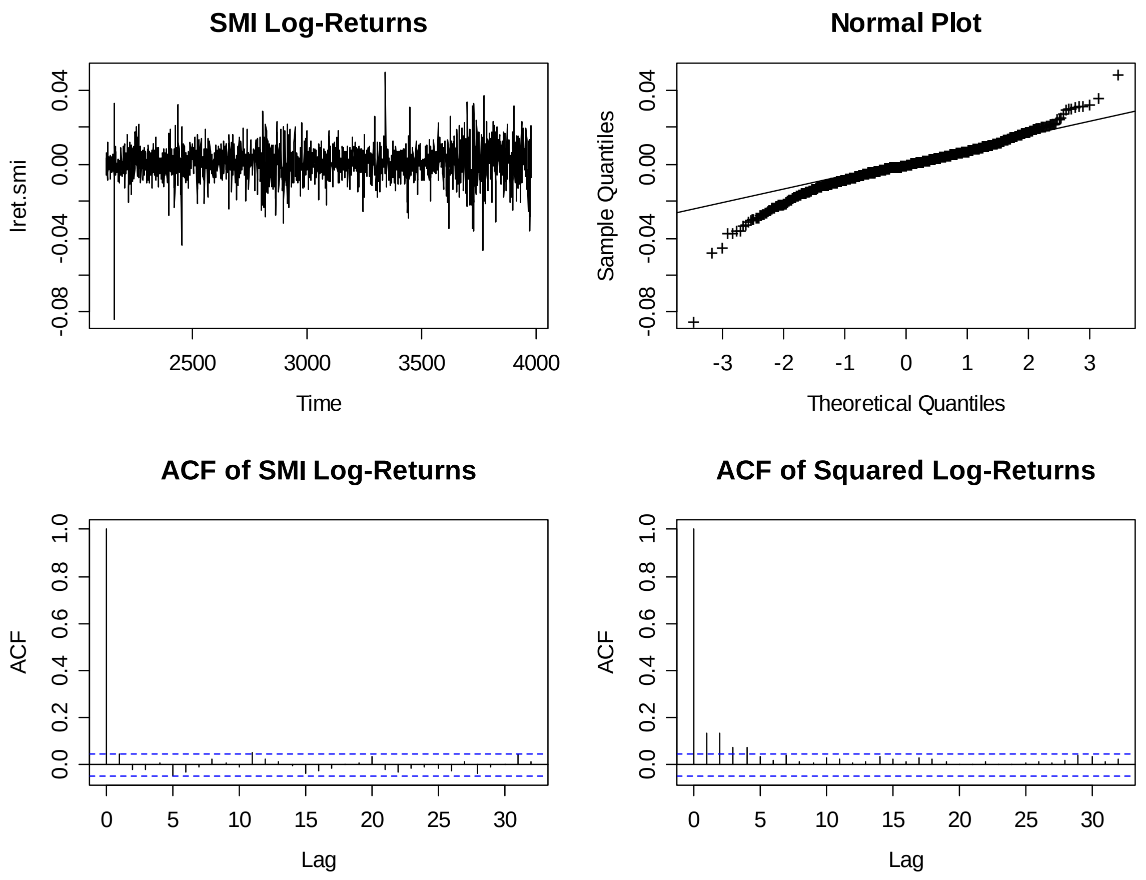
5) Check the accuracy of the fitted model by residual analysis. These must look like white noise. If thus far, there is ambiguity in the model order, AIC analysis can serve to come to a final decision.

Next, we turn our attention to series that have neither trend nor seasonality, but show serially dependent variance.

7.3 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 variance, 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 variation, and thus the variance of the series is serially correlated, a phenomenon that is called *conditional heteroskedasticity*. This is a violation of the stationarity assumption, and thus, 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:



7.3.1 The ARCH and GARCH Models

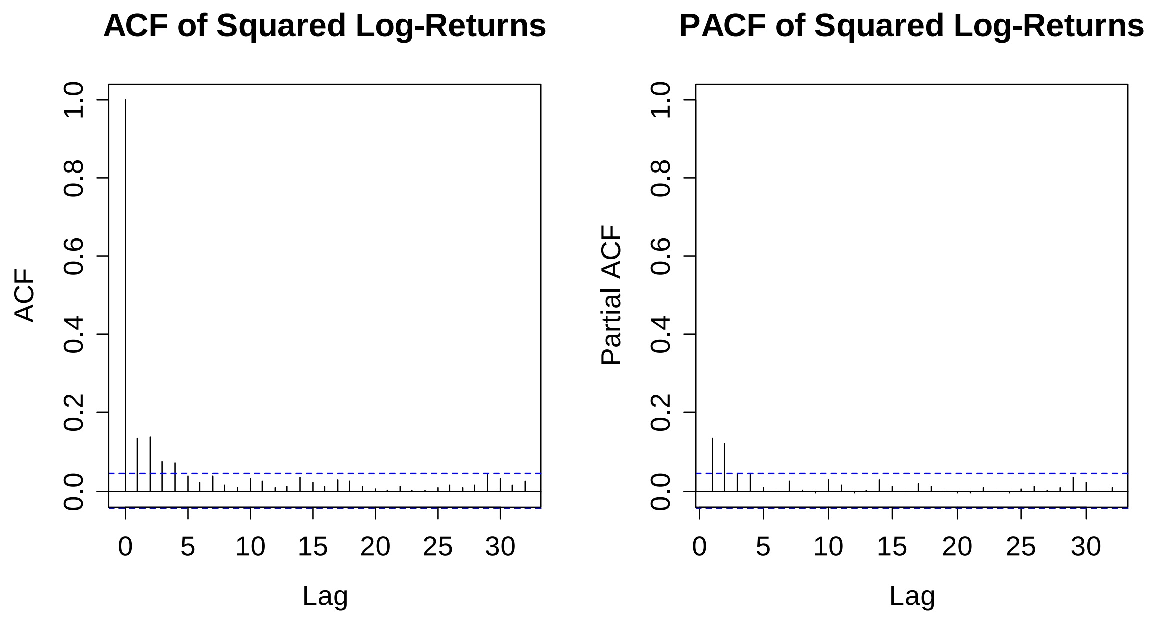
In order to account for volatility, we require a model that allows for conditional changes in the variance. The most simple and intuitive way of doing this is to use an autoregressive model for the variance process. Thus, a series E_t is first-order autoregressive conditional heteroskedastic, denoted 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 above, but the 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(q,p) model does. A series $E_t = W_t \sqrt{H_t}$ is GARCH(q,p) if:

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

7.3.2 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.

8 Forecasting

One of the principal goals with time series analysis is to produce predictions which show the future evolution of the data. This is what it is: an extrapolation in the time domain. And as we all know, extrapolation is always (at least slightly) problematic and can lead to false conclusions. Of course, this is no different with time series forecasting.

The saying is that the task we are faced with 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 and has a few gentle bends only, things get more complicated as soon as there are some sharp and unexpected bends in the road. 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 the above, one might conclude that the principal source of uncertainty is inherent in the process, i.e. comes from the innovations. However, in practice, this is usually different, and several other factors can threaten the reliability of any forecasting procedure. In particular:

- We need to be certain that the data generating process does not change over time, i.e. continues in the future as it was observed in the past.
- 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.
- Even if we are so lucky to find the correct data-generating process (or in cases we know it), there is additional uncertainty arising from the estimation of the parameters.

Keeping these general warnings 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 predicted. 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 4.2, the decomposition approach for non-stationary time series helps a great deal for visualization and modeling. 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()` procedure in **R**, it is very popular and often used in applied sciences.

8.1 Forecasting ARMA

We suppose that we are given a time series, for which an appropriate AR, MA or ARMA 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 involve either involve the past observations, and/or the 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.

8.1.1 Forecasting AR(1)

For simplicity, we first consider a mean-zero, stationary AR(1) process with model equation:

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

where 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 forecast the next instance of a time series with the observed value of the previous one, in particular:

$$\hat{X}_{n+1,1:n} = \alpha_1 x_n \, .$$

For the k-step forecast with $k > 1$, we just need to repeatedly plug-in the model equation, and as use the fact that the conditional expectation of the innovations is zero:

$$\begin{aligned}
\hat{X}_{n+k,1:n} &= E[X_{n+k} \mid X_1,...,X_n] \\
&= E[\alpha_1 X_{n+k-1} + E_{n+k} \mid X_1,...,X_n] \\
&= \alpha_1 E[X_{n+k-1} \mid X_1,...,X_n] \\
&= ... \\
&= \alpha_1^k x_n
\end{aligned}$$

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. 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 any desired horizon.

As always, a prediction is much more useful in practice if one knows how precise it is. 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:

$$\alpha_1 x_n \pm 1.96 \cdot \sigma_E .$$

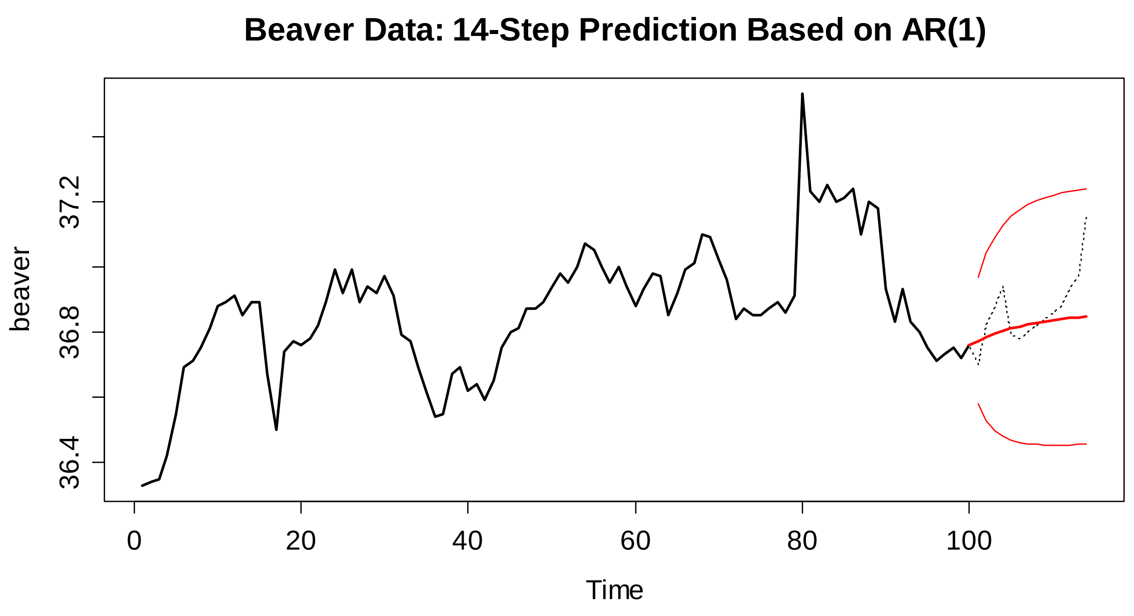
Again, for practical implementation, we need to plug-in $\hat{\alpha}_1$ and $\hat{\sigma}_E$. For a k-step prediction, the 95% prognosis interval is:

$$\alpha_1 x_n \pm 1.96 \cdot \left(1+\sum_{j=1}^{k-1} \alpha_1^{2j}\right) \cdot \sigma_E .$$

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 process variance.

Practical Example

We now turn our attention to a practical example, where we apply the **R** functions which implement the above theory. This is the Beaver data we had already discussed in section 4.3.3. An AR(1) model is appropriate, and for estimating the coefficients, we omit the last 14 observations from the data. These will be predicted, and the true values are needed for verifying the prediction.



The **R** commands for fitting the model on the training data and producing the 14-step prediction are as follows:

```
> btrain   <- window(beaver, 1, 100)
> 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 standard error, respectively. We now turn our attention to how the forecast is visualized:

```
> plot(beaver, lty=3)
> lines(btrain, lwd=2)
> lines(pred$pred, lwd=2, col="red")
> lines(pred$pred+pred$se*1.96, col="red")
> lines(pred$pred-pred$se*1.96, col="red")
```

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 quick. Because the estimated coefficient is $\hat{\alpha}_1 = 0.87$, and thus relatively close to one, the approach still takes some time.

8.1.2 Forecasting AR(p)

Forecasting from AR(p) processes works based on the same principles as explained above for the AR(1), i.e. we use the conditional expected value. The algebra for writing the forecasting formulae is somewhat more laborious, but not more difficult. Thus, we do without displaying it here, and directly present the 1-step-forecast:

$$\hat{X}_{n+1,1:n} = \alpha_1 x_n + \alpha_2 x_{n-1} + ... + \alpha_p x_{n-p}$$

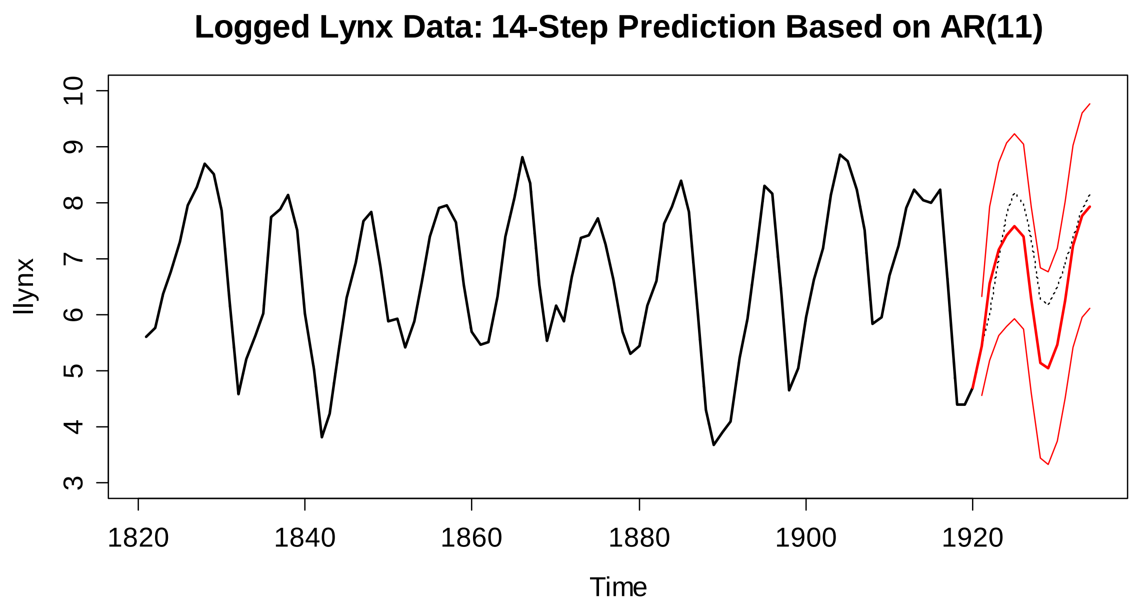
The question is, what do we do for longer forecasting horizons? There, the forecast is again based on the linear combination of the p past instances. For the ones with an index between 1 and n, the observed value x_t is used. Else, if the index exceeds n, we just plug-in the forecasted values $\hat{x}_{t,1:n}$. Thus, the general formula is:

$$\hat{X}_{n+k,1:n} = \alpha_1 \hat{X}_{n+k-1,1:n} + ... + \alpha_p \hat{X}_{n+k-p,1:n} ,$$

where $\hat{X}_{t,1:n} = x_t$ in all cases where $t \le n$, i.e. an observed value is available.

Practical Example

We consider the 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. Also, we do without showing the R code, because it is identical to the one from the previous example.



We observe that the forecast tracks the general behavior of the series well, though the level of the series is underestimated somewhat. This is, however, not due to an "error" of ours, it is just that the values were higher than the past observations suggested. We finish this section with some remarks:

- Forecasting from an AR(p) only requires knowledge about the last p instances of the series, plus the model parameters $\alpha_1,...,\alpha_p$ and the global series mean μ. Earlier values of the series are not required, the model thus has a Markov property of order p.

- The prediction intervals only account for the uncertainty caused by the innovation variance, but not for the one caused by model misconception, and the plug-in of estimated parameters. Thus in practice, a true 95% interval would most likely be wider than shown above.

8.1.3 Forecasting MA(1)

We here consider an invertible MA(1) process, where the model equation is as follows:

$$X_t = E_t + \beta_1 E_{t-1},$$

where 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 forecast 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 thus generally is different from zero. Moreover, 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(∞). 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 $\le q$. These can be mitigated, if we again condition on the infinite past of the process.

$$\begin{aligned} \hat{X}_{n+1,1:n} &= E[X_{n+1} \mid X_{-\infty}^{n}] \\ &= \sum_{i=1}^{p} \alpha_i E[X_{n+1-i} \mid X_{-\infty}^{n}] + E[E_{n+1} \mid X_{-\infty}^{n}] + \sum_{j=1}^{q} \beta_j E[E_{n+1-j} \mid X_{-\infty}^{n}] \\ &= \sum_{i=1}^{p} \alpha_i x_{n+1-i} + \sum_{i=1}^{q} \beta_j E[E_{n+1-j} \mid X_{-\infty}^{n}] \end{aligned}$$

If we are aiming for k -step forecasting, we can use a recursive prediction scheme:

$$\hat{X}_{n+k,1:n} = \sum_{i=1}^{p} \alpha_i E[X_{n+k-i} \mid X_{-\infty}^{n}] + \sum_{j=1}^{q} \beta_j E[E_{n+k-j} \mid X_{-\infty}^{n}],$$

where for the AR- and MA-part the conditional expectations are:

$$E[X_t \mid X_{-\infty}^{n}] = \begin{cases} x_t, & if \ \ t \le n \\ \hat{X}_{t,1:n}, & if \ \ t > n \end{cases}$$

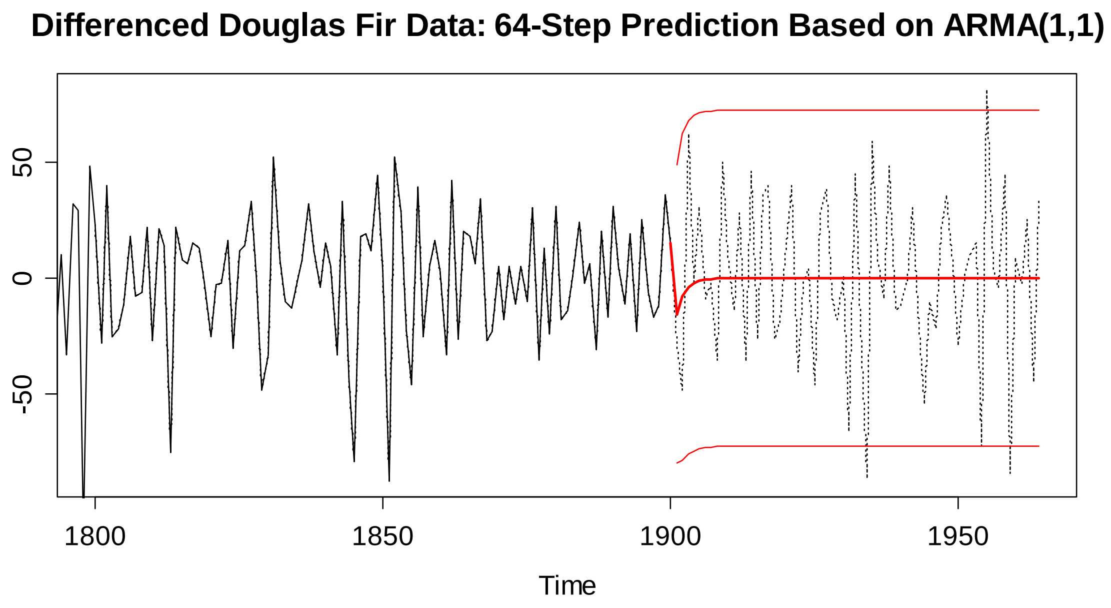
$$E[E_t \mid X_{-\infty}^{n}] = \begin{cases} e_t, & if \ \ t \le n \\ 0, & if \ \ t > n \end{cases}$$

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 the year rings over a period from 1107 to 1964. Because the data are non-stationary, we take differences and model these. An ARMA(1,1) seems appropriate. We put the last 64 observations aside so that we can verify our predictions. Then, the model is fitted and the Box-Jenkins predictions are obtained. The result, including a 95% prognosis interval, is shown below.



We observe that the forecast goes to zero exponentially quickly. However, it is in fact different from zero for all times. Moreover, all observations down to the very first one are used for obtaining the predictions. Again, the ARMA model combines the properties from pure AR and MA processes.

8.2 ARIMA/SARIMA

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 mean:

$$a_t = \alpha x_t + (1-\alpha)x_{t-1}, \text{ with } 0 < \alpha < 1.$$

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.

$$\hat{X}_{n+k,1:n} = a_n, \text{ for all } k = 1,2,....$$

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:

$$a_t = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + ...$$

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 $t = 2,3,...$. 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 $SS1PE$:

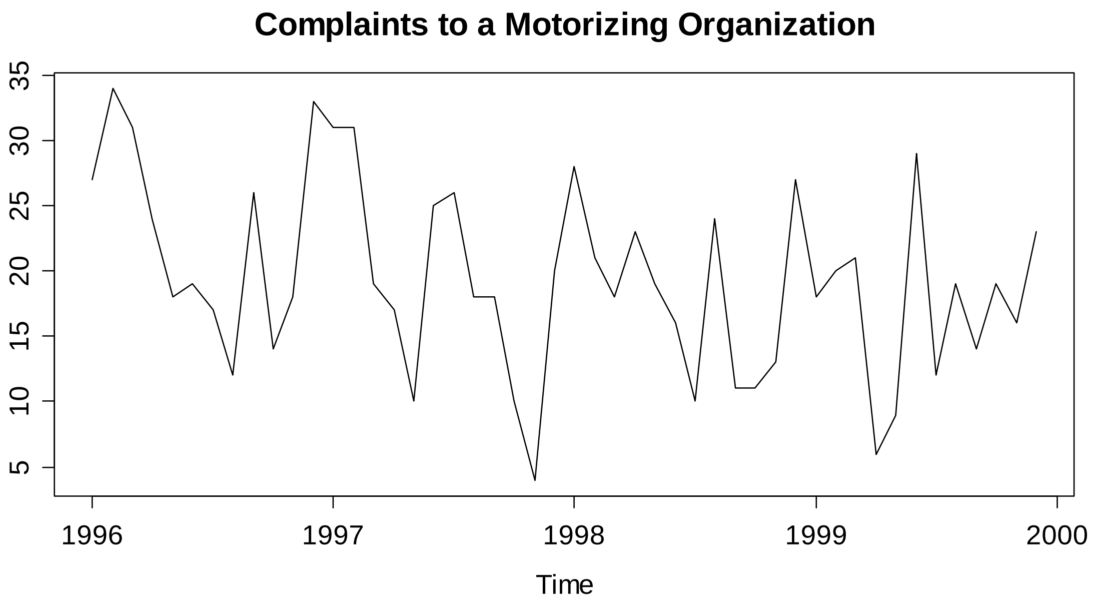
$$SS1PE = \sum_{t=2}^{n} e_t^2\,.$$

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:

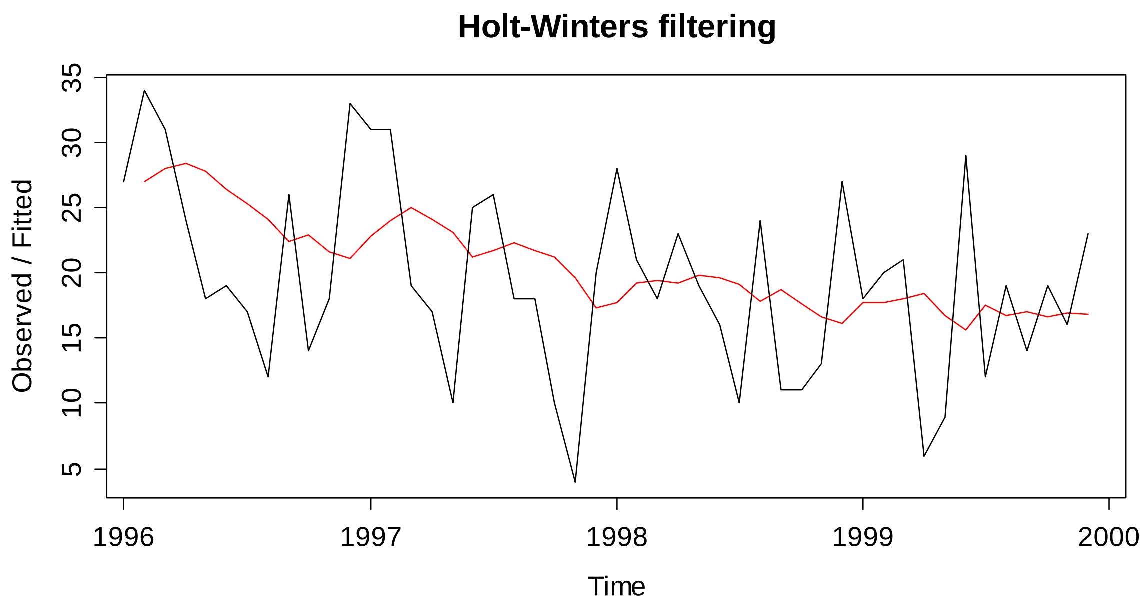
```
> www  <- "http://www.massey.ac.nz/~pscowper/ts/motororg.dat"
> dat  <- read.table(www, header=TRUE)
> cmpl <- ts(dat$complaints, start=c(1996,1), freq=12)
> plot(cmpl, ylab="", main="Complaints ...")
```



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
Holt-Winters exponential smoothing without trend and without
seasonal component.

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 now three smoothing parameters α, β, γ which are aimed at level, slope and season. The explanation of these 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 appropriate seasonal effect 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:

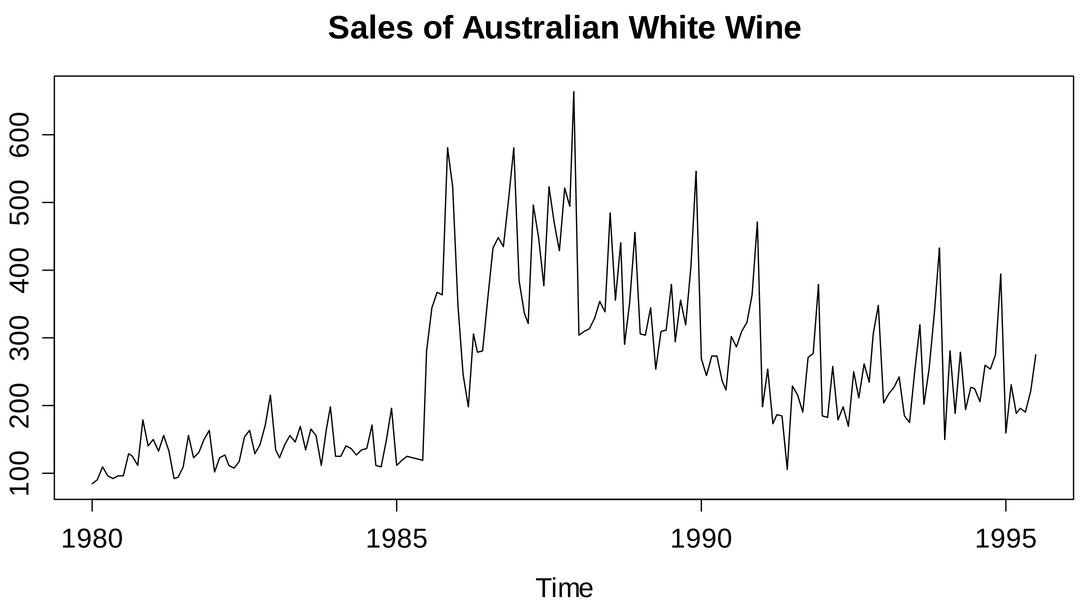
$$\hat{X}_{n+k,1:n} = a_n + kb_n + s_{n+k-p} \, ,$$

i.e. the current level with linear trend extrapolation plus the appropriate seasonal effect. The following practical example nicely illustrates the method.

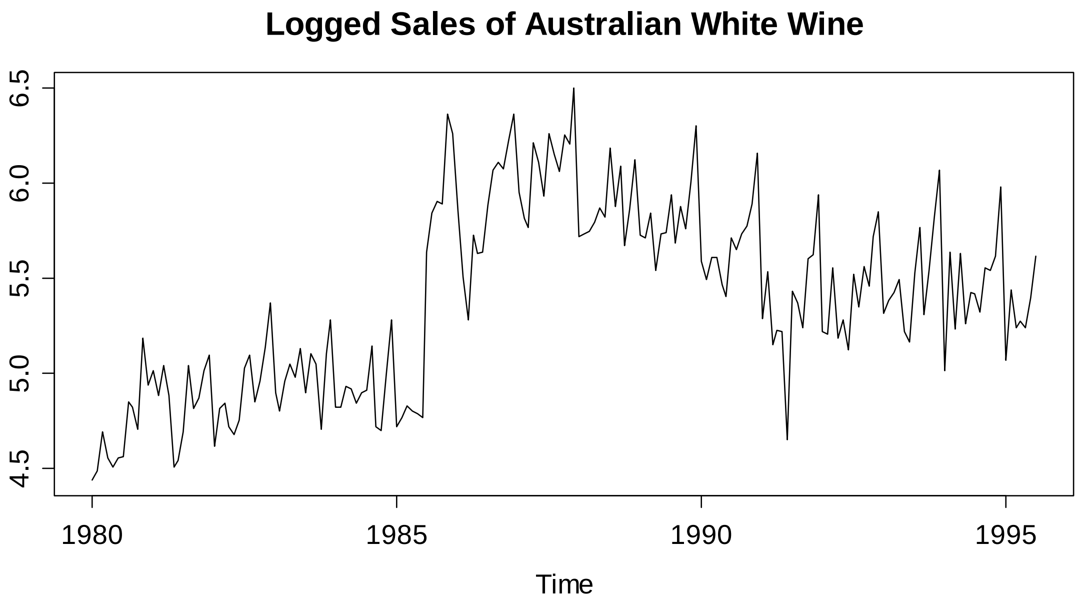
Practical Example

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://www.massey.ac.nz/~pscowper/ts/wine.dat"
> dat  <- read.table(www, header=TRUE)
> 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 ...")
```

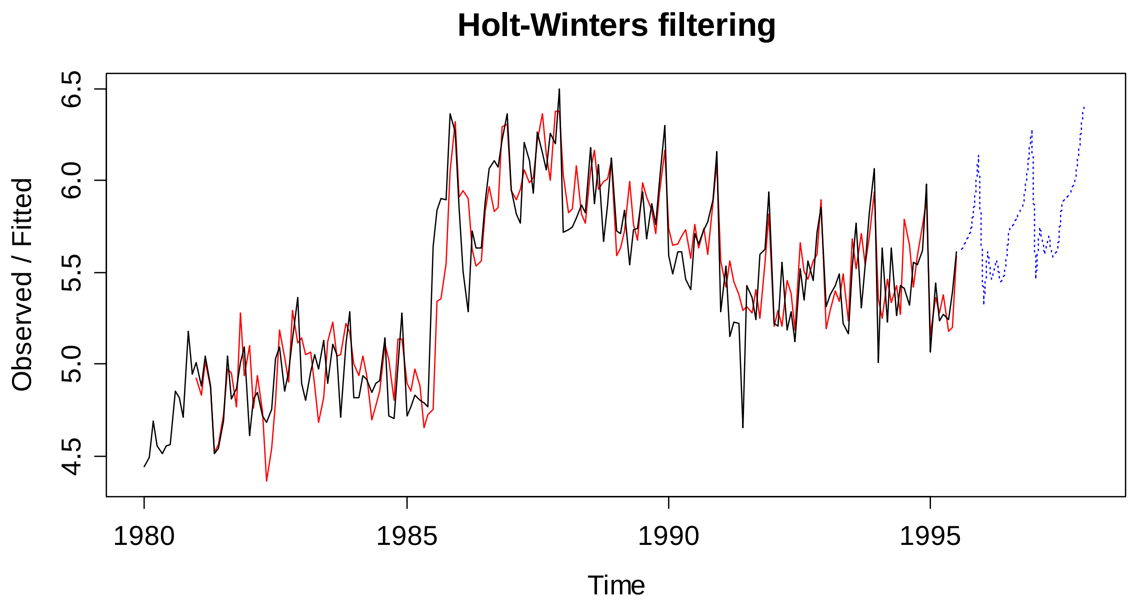


The transformation seems successful, thus we proceed to the Holt-Winters modeling. When we apply parameter estimation by $SS1PE$, this is straightforward. The fit contains the current estimates for level, trend and seasonality. Note that these are only valid for time n, and not for the entire series. 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
Holt-Winters exponential smoothing with trend and additive
seasonal component.

Call:
 HoltWinters(x = log(aww))

Smoothing parameters:
 alpha:  0.4148028
 beta :  0
 gamma:  0.4741967

Coefficients:
a     5.62591329
b     0.01148402
s1   -0.01230437
s2    0.01344762
s3    0.06000025
s4    0.20894897
s5    0.45515787
s6   -0.37315236
s7   -0.09709593
s8   -0.25718994
s9   -0.17107682
s10  -0.29304652
s11  -0.26986816
s12  -0.01984965
```

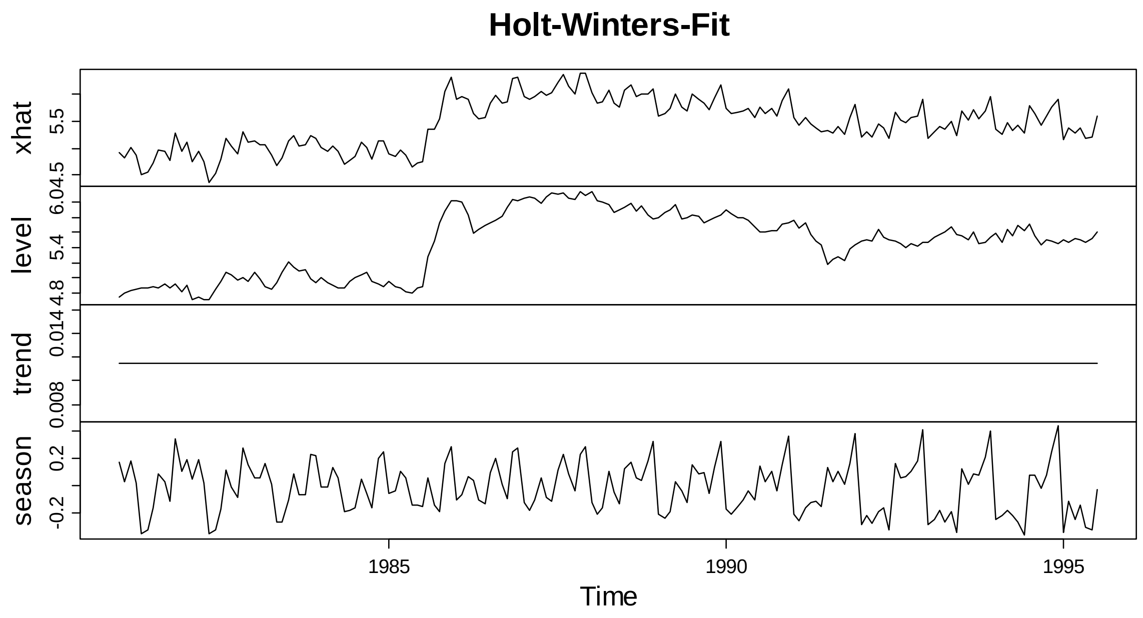
The coefficient values (at time n) are also the ones which are used for forecasting from that series with the formula given above. We produce a prediction up until the end of 1998, which is a 29-step forecast. The R commands are:

```
> plot(fit, xlim=c(1980, 1998))
> lines(predict(fit, n.ahead=29), col="blue", lty=3)
```



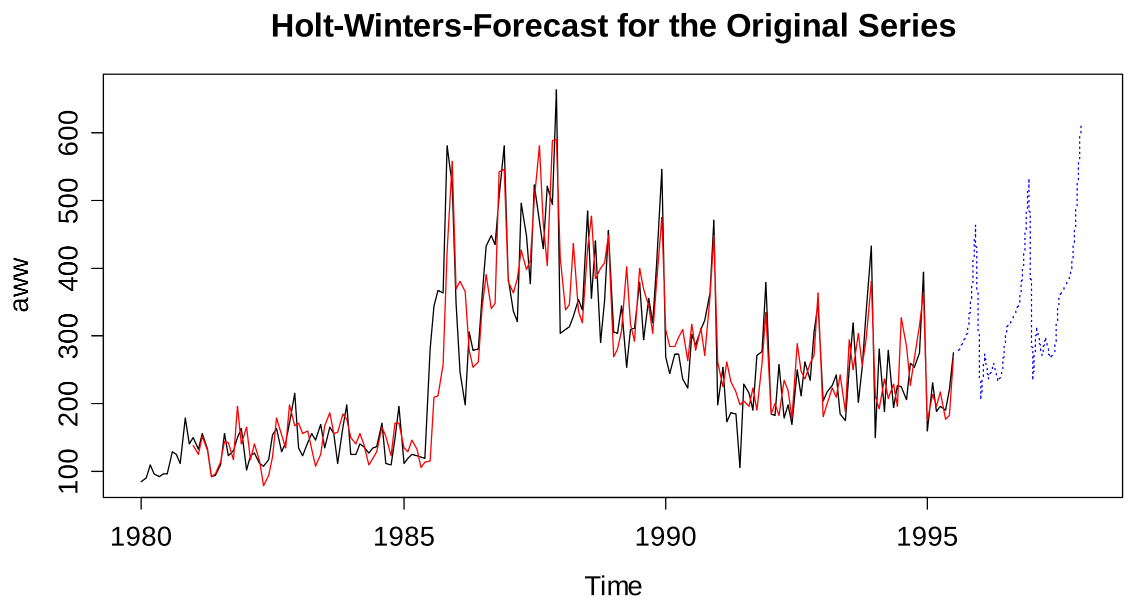
It is also very instructive to plot how level, trend and seasonality evolved over time. This can be done very simply in **R**:

```
> plot(fit$fitted, main="Holt-Winters-Fit")
```



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.

8.4 Forecasting Decomposed Series

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.

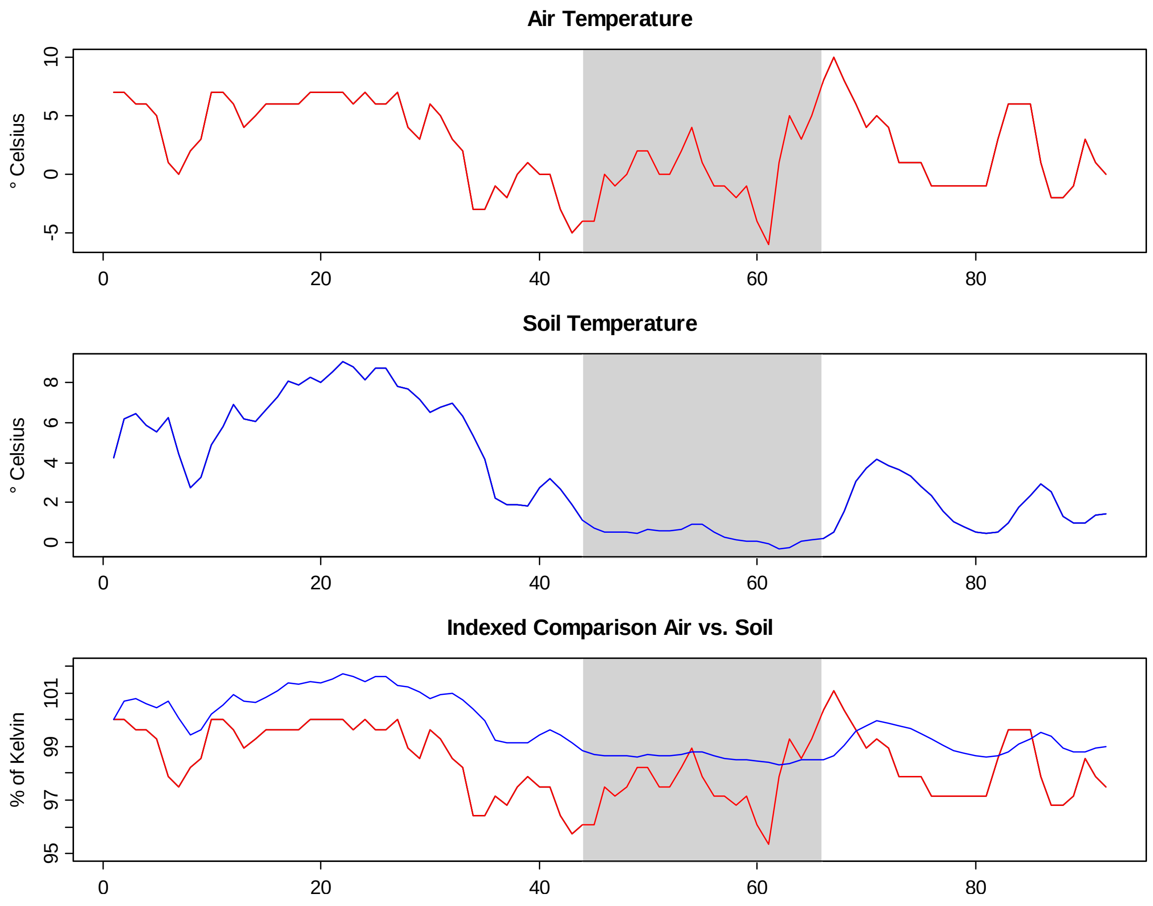
9.1 Practical Example

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.

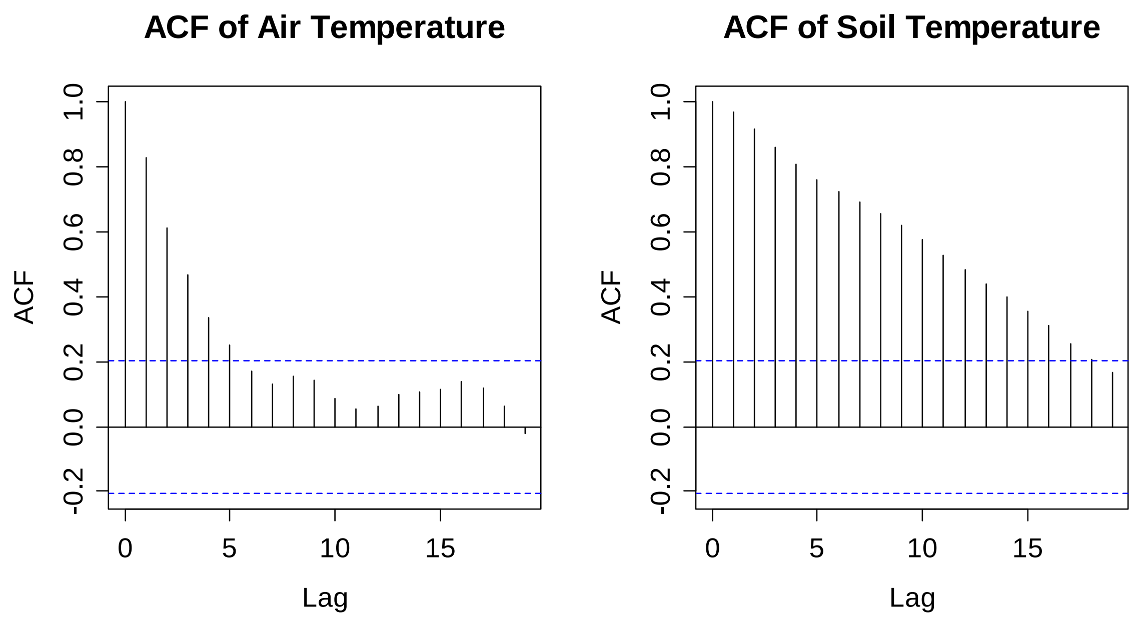
Ambient air temperatures influence ground temperatures with a certain temporal delay. Borehole temperatures measured at 0.5m depth in alpine permafrost terrain, as well as air temperatures measured at or nearby the boreholes will be used to model this dependency. The reaction of the ground on the air 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.

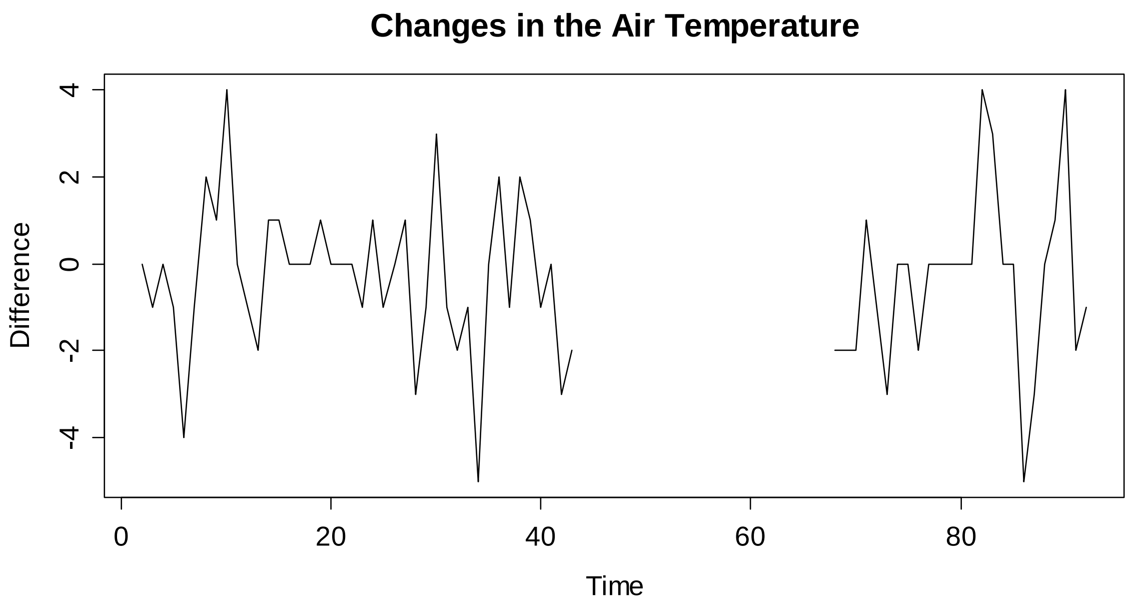
Because the snow insulates the ground, we do not expect the soil to follow the air temperature during that period. Hence, we set all values during that period equal to NA. The time series plots, and especially the indexed plot where both series are shown, clearly indicate that the soil temperature reacts 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.

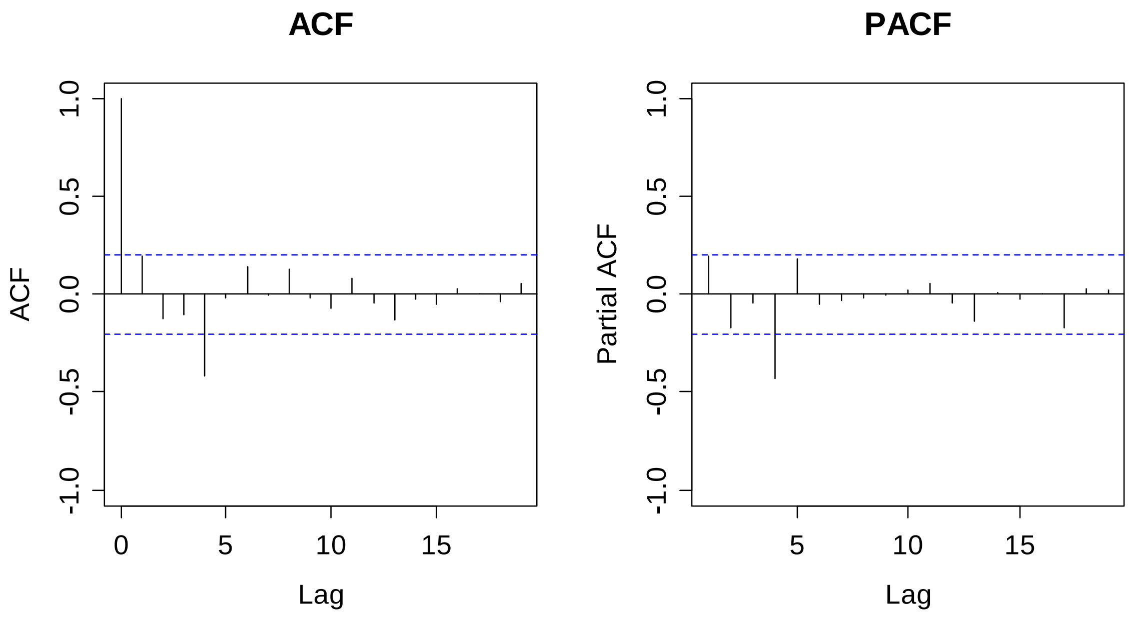


As we had stated above, multivariate time series analysis requires stationarity. Is this met with our series? The time series plot does not give a very clear answer. Science tells us that temperature has a seasonal pattern. Moreover, the correlogram of the two series is enlightening.

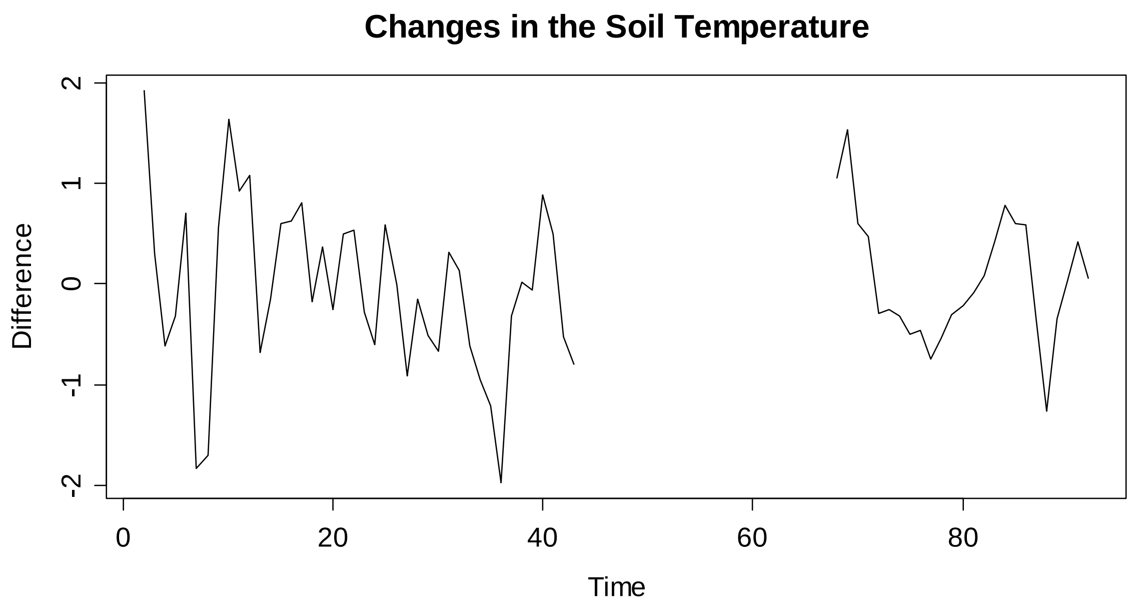


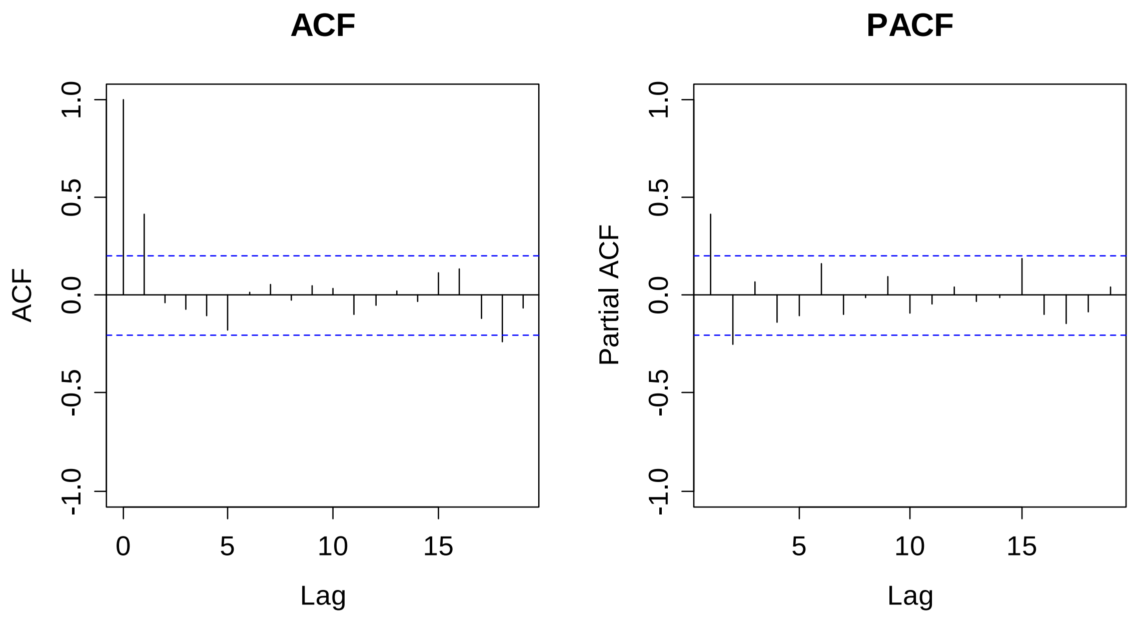
The ACF exhibits a slow decay, especially for the soil temperature. Thus, we decide to perform lag 1 differencing before analyzing the series. This has another advantage: we are then exploring how changes in the air temperature are associated with changes in the soil temperature and if so, what the time 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. like white noise.

For the changes in the soil temperature, model search is easier. ACF and PACF suggest either a MA(1), an ARMA(2,1) or an AR(2). From these three models, the first one, MA(1) shows both the lowest AIC value as well as the “best looking” residuals.

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:

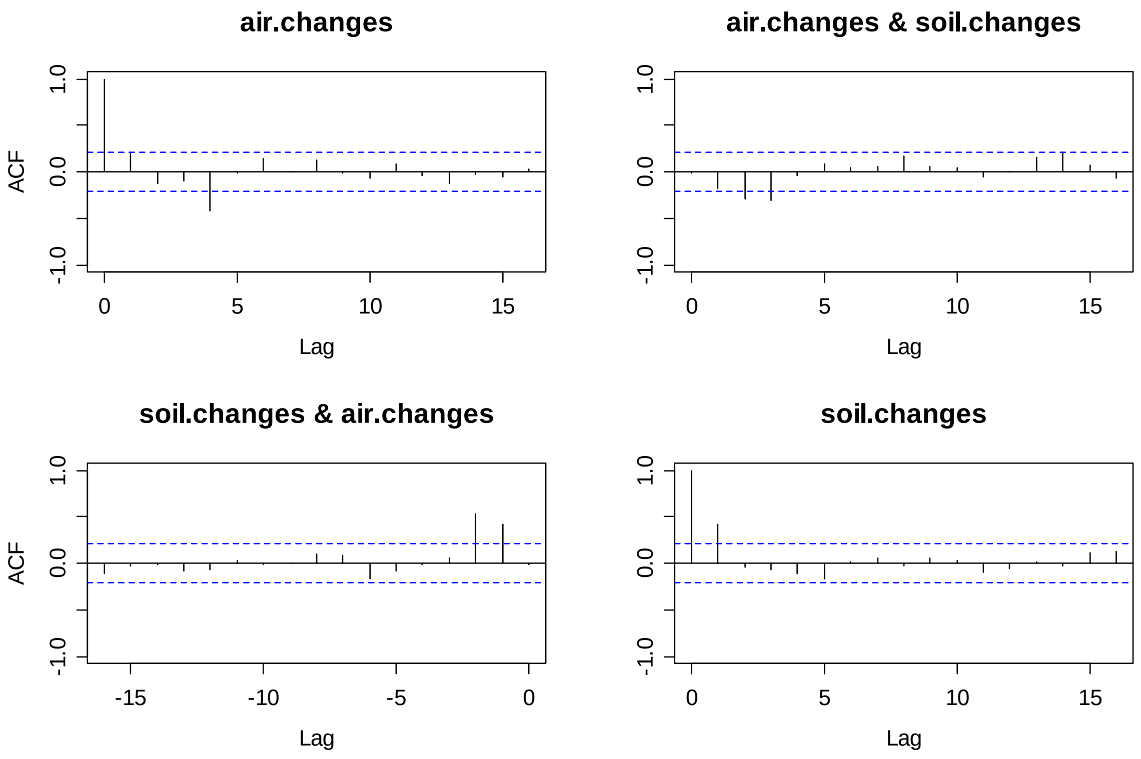
$$\hat{\gamma}_{12}(k) = \frac{1}{n}\sum_t (x_{1,t+k} - \bar{x}_1)(x_{2,t} - \bar{x}_2) \text{ and } \hat{\gamma}_{21}(k) = \frac{1}{n}\sum_t (x_{2,t+k} - \bar{x}_2)(x_{1,t} - \bar{x}_1),$$

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.

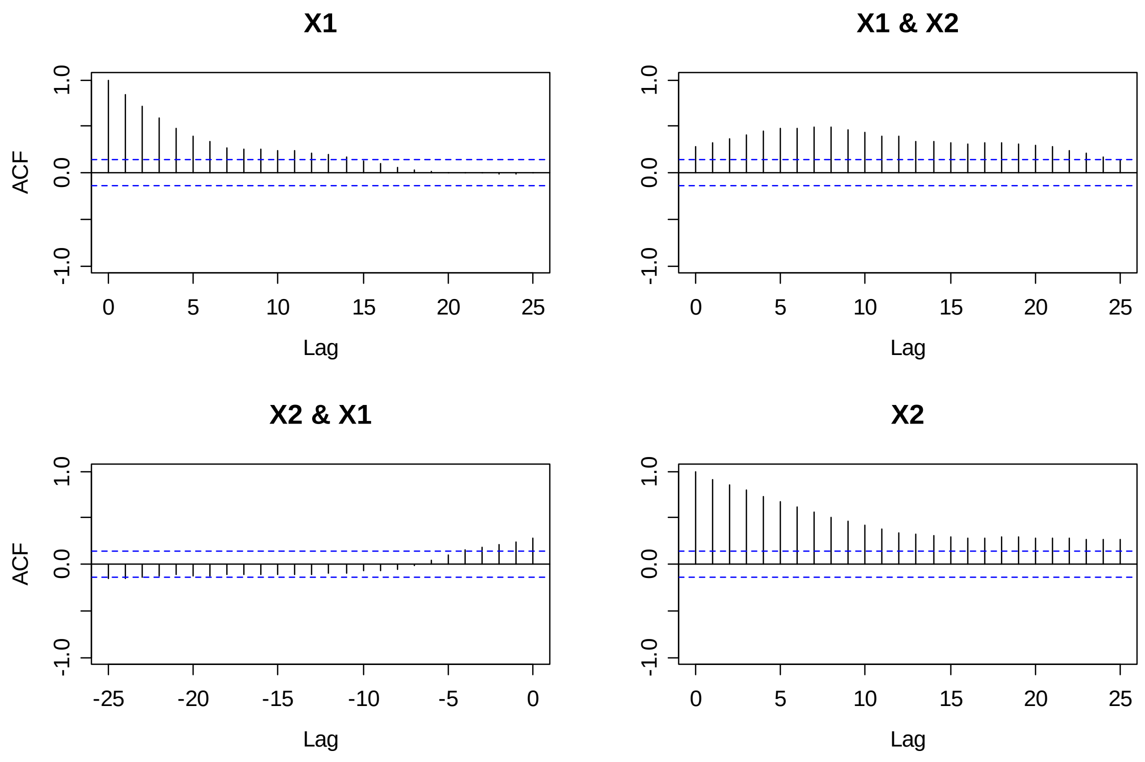
Case 3: No cross correlations for all lags and one series uncorrelated

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.

to get an approximate 95% confidence interval that helps to decide whether an estimated cross-correlation is only randomly different from 0.



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(∞) representation. For the cross-correlations between U_t and V_t and between X_t and Y_t, the following relation holds:

$$\rho_{UV}(k) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} a_i b_j \rho_{X_1X_2}(k+i-j)$$

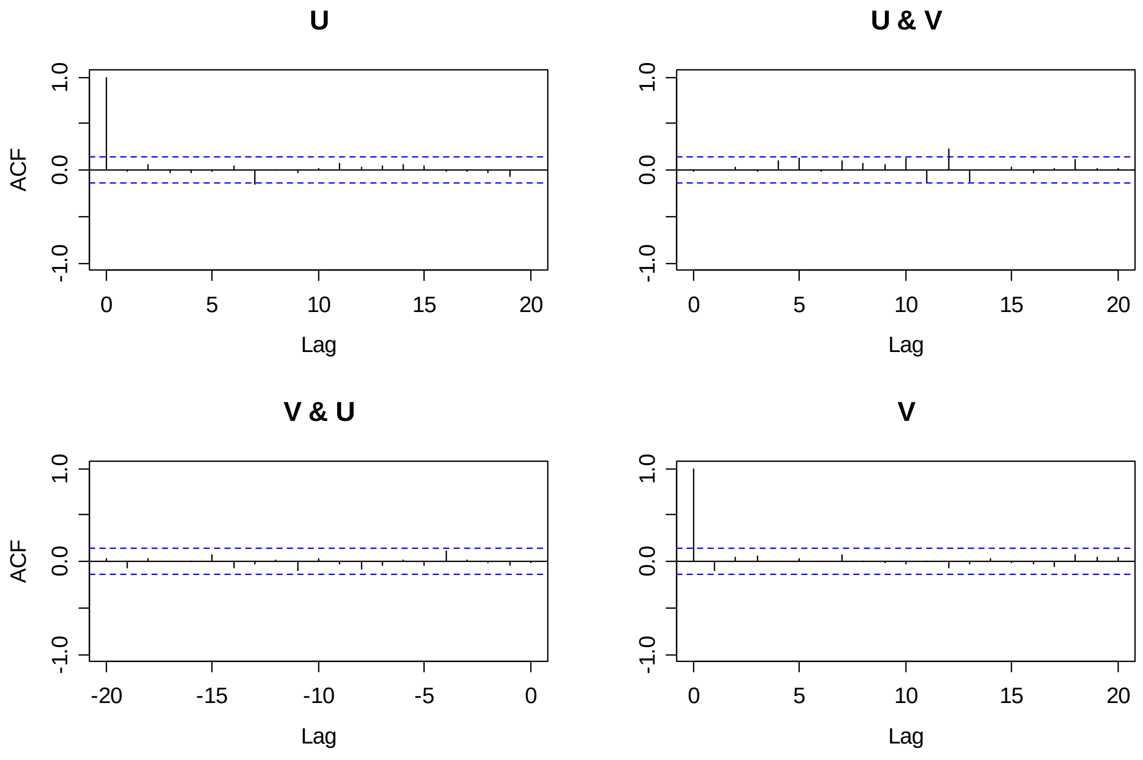
We conjecture that for two independent processes X_1 and X_2, where all cross correlation coefficients $\rho_{X_1X_2}(k)=0$, also all $\rho_{UV}(k)=0$. Additionally, the converse is also true, i.e. it follows from "U_t and V_t uncorrelated" that the original processes X_1 and X_2 are uncorrelated, too. 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 $\hat{\alpha}_2 = 0.917$.



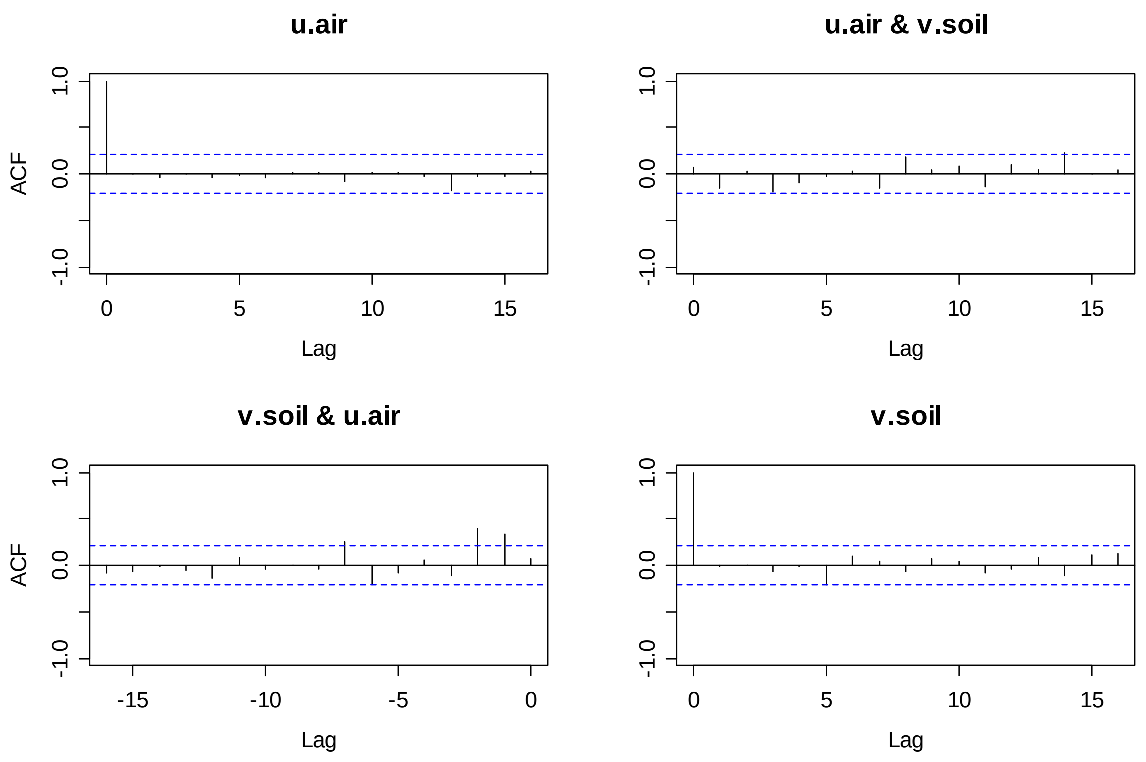
The figure on the previous page shows both the auto and cross correlations of the prewhitened series. We emphasize again that we here consider the residuals from the AR(1) models that were fitted to series X_1 and X_2. We observe that, as we expect, there are no significant autocorrelations, and there 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 soil temperature 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:

$$Z_t = \sum_{j=0}^{\infty} \nu_j D_{t-j} + U_t,$$

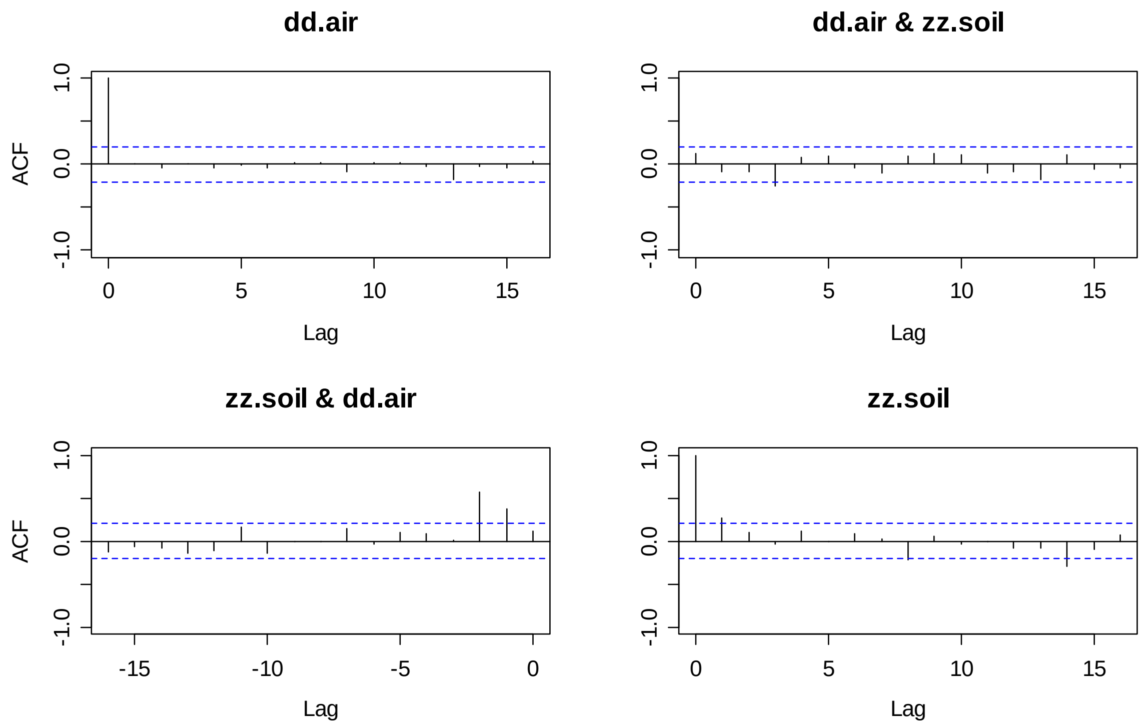
where the coefficients ν_j are identical than for the previous formulation of the model. The advantage of this latest formulation, however, is that the input series D_t is now white noise, such that the above special case applies, and the transfer function model coefficients can be obtained by a straightforward 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 temperature changes and the soil temperature changes that had been filtered with the air's 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.

10 Spectral Analysis

During this course, we have encountered several time series which show periodic behavior. Prominent examples include the number of shot lynx in the Mackenzie River district in Canada, as well as the wave tank data from section 4.3. In these series, the periodicity is not deterministic, but stochastic. An important goal is to understand the cycles at which highs and lows in the data appear.

In this chapter, we will introduce *spectral analysis* as a descriptive means for showing the character of, and the dependency structure within a time series. This will be based on interpreting the series as a superposition of cyclic components, namely as a *linear combination of harmonic oscillations*. We will introduce the *periodogram*, where the aim is to show which frequencies contribute most importantly to the variation in the series.

In spirit, such an analysis is related to the correlogram. In fact, one can show that the information in the correlogram and the periodogram are mathematically equivalent. However, the two approaches still provide different, complementary views on a series and it is thus often worthwhile to pursue both approaches. Finally, we here also mention that in some areas time series are preferably analyzed in the time domain, whereas in other applied fields, e.g. electrical engineering, geophysics and econometrics, the frequency approach predominates.

10.1 Decomposing in the Frequency Domain

We will here first introduce some background and theory on how to decompose time series into cyclic components and then lay the focus on the efficient estimation of these.

10.1.1 Harmonic Oscillations

The simplest and best known periodic functions are sine and cosine. It is thus appealing to use these as a basis for decomposing time series. A harmonic oscillation is of the form

$$y(t) = a \cdot \cos(2\pi\nu t - \phi) .$$

Here, we call a the amplitude, ν is the frequency and ϕ is the phase. Apparently, the function $y(t)$ is periodic, and the period is $T = 1/\nu$. It is common to write the above harmonic oscillation in a different form, i.e.:

$$y(t) = \alpha \cdot \cos(2\pi\nu t) + \beta \cdot \sin(2\pi\nu t) ,$$

where in fact $\alpha = a\cos(\phi)$ and $\beta = a\sin(\phi)$. The advantage of this latter form is that if we want to fit a harmonic oscillation with fixed frequency to data, which means estimating amplitude and phase, we face a linear problem instead of a non-linear one, as it was the case in the previous formulation. The time can be either continuous or discrete. In the context of our analysis of discrete time series, only the latter will be relevant.

Now, if fitting a harmonic oscillation to discrete data, we face an identification problem: If frequency ν fits, then all higher frequencies such as $\nu+1, \nu+2, ...$ will fit as well. This phenomenon is now as aliasing. The plot below shows harmonics where $a=1$ and $\phi=0$. As frequencies, we choose both $\nu=1/6$ and $\nu=1+1/6$. We observe that we cannot decide upon which of the two frequencies generated our discrete time observations. Naturally, the time resolution of our series determines which frequencies we can identify. Or more clearly: we take the point that our data do not allow to identify periodicities with frequency $\nu>1/2$, i.e. that harmonics which oscillate more than once between two observations.

10.1.2 Superposition of Harmonics

In a real-world stationary time series, it is rare to inexistent that only one single periodicity that can be attributed to a single frequency makes up for all the variation that is observed. Thus, for a decomposition of the series into a number of periodicities with different frequency, we choose the following regression-type approach:

$$X_t = \alpha_0 + \sum_{k=1}^{m} (\alpha_k \cos(2\pi\nu_k t) + \beta_k \sin(2\pi\nu_k t)) + E_t ,$$

where α_k, β_k are interpreted as the unknown parameters, E_t is an iid error term with expectation zero and $\nu_1, ..., \nu_m$ is a set of pre-defined frequencies. Under these assumptions, we can obtain estimates $\hat{\alpha}_k, \hat{\beta}_k$ with the ordinary least squares algorithm. As for the frequencies, we choose multiples of $1/n$, i.e.

$\nu_k = k/n$, for $k=1,...,m$ with $m=\lfloor n/2 \rfloor$.

These are called the *Fourier frequencies*. Using some mathematics, one can prove that the above regression problem has orthogonal design. Thus, the estimated coefficients $\hat{\alpha}_k, \hat{\beta}_k$ are uncorrelated and (for $k>0$) have variance $2\sigma_E^2/2$. Because we are also spending n parameters for the n observations, the frequency decomposition model fits perfectly, i.e. all residuals are zero. Another very important result is that the

sum of squared residuals $\sum_{i=1}^{n} r_i^2$ increases by $\frac{n}{2}(\hat{\alpha}_k^2 + \hat{\beta}_k^2)$

if the frequency ν_k is omitted from the model. We can use this property to gauge the prominence of a particular frequency in the decomposition model. This is what is done with the periodogram, which we will discuss in detail in the following section.

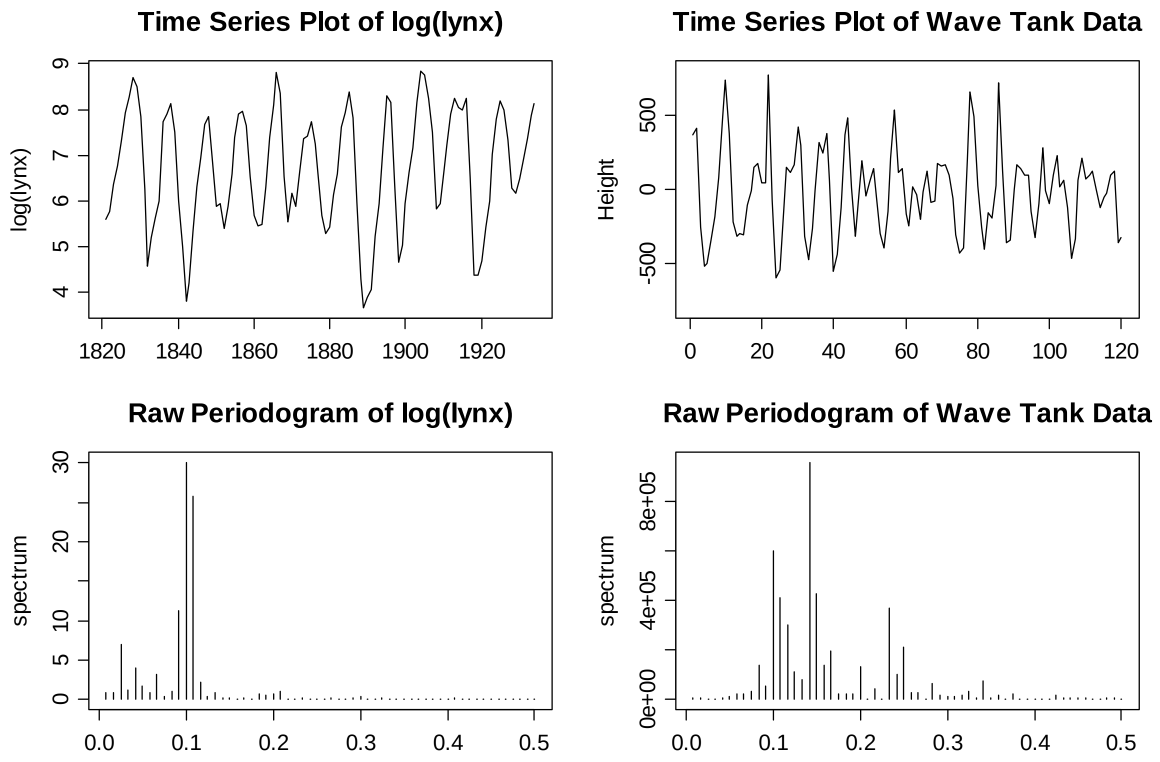
10.1.3 The Periodogram

The periodogram quantifies the presence of periodicities in a time series. It is based on half of the increase in sum of squared residuals in the decomposition model if a particular frequency is omitted. We can rewrite that directly as a function of the observations:

$$\begin{aligned} I_n(\nu_k) &= \frac{n}{4}(\hat{\alpha}_k^2 + \hat{\beta}_k^2) \\ &= \frac{1}{n}\left(\sum_{t=1}^{n} x_t \cos(2\pi\nu_k t)\right)^2 + \frac{1}{n}\left(\sum_{t=1}^{n} x_t \sin(2\pi\nu_k t)\right)^2 \end{aligned}$$

The result is then plotted versus the frequency ν_k, and this is known as the raw periodogram. In R, we can use the convenient function `spec.pgram()`. We illustrate its use with the lynx and the wave tank data:

```
> spec.pgram(log(lynx), log="no", type="h")
> spec.pgram(wave, log="no", type="h")
```



The periodogram of the logged lynx data is easy to read: the most prominent frequencies in this series with 114 observations are the ones near 0.1, more exactly, these are $\nu_{11} = 11/114 = 0.096$ and $\nu_{12} = 12/114 = 0.105$. The period of these frequencies is $1/\nu_k$ and thus, $114/11 = 10.36$ and $114/12 = 9.50$. This suggests that the series shows a peak at around every 10th observation which is clearly the case in practice. We can also say that the highs/lows appear between 11 and 12 times in the series. Also this can easily be verified in the time series plot.

Then, there is a secondary peak at $\nu_3 = 3/114$. This must be a cyclic component that appears three times in our data, and the period is $114/3 = 38$. Thus, these are the 40-year-superhighs and -lows that we had identified already earlier.

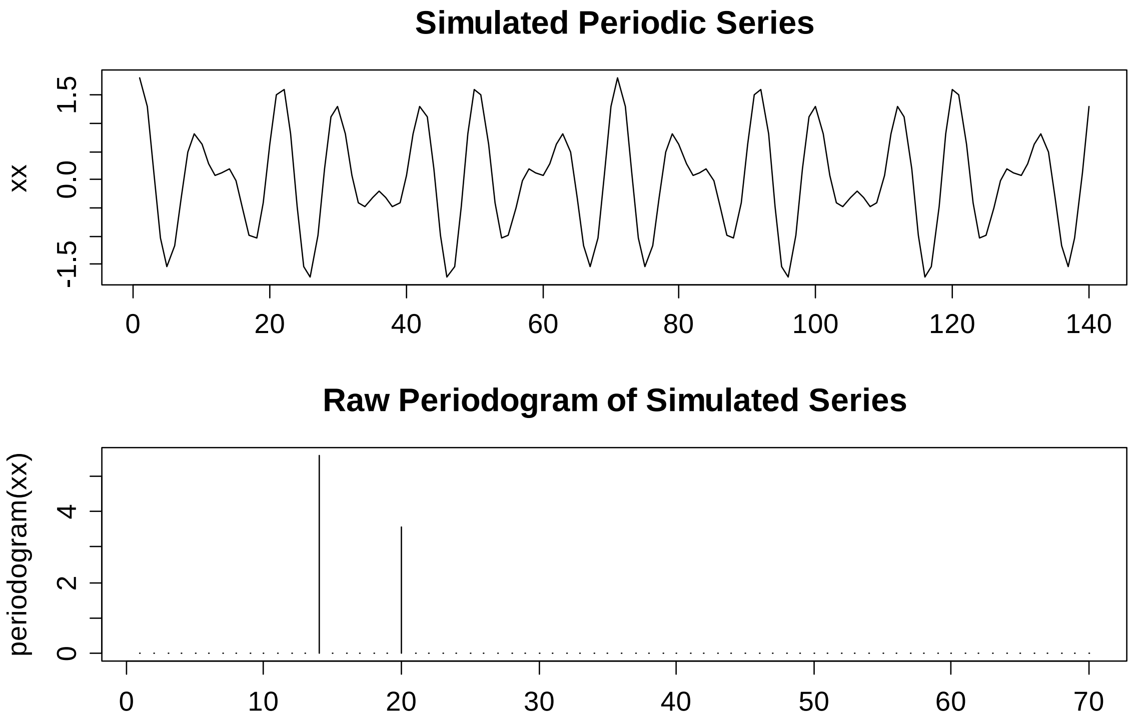
For the wave tank data, we here consider the first 120 observations only. The periodogram is not as clean as for the logged lynx data, but we will try with an interpretation, too. The most prominent peaks are at $k = 12, 17$ and 30. Thus we have a superposition of cycles which last 4, 7 and 10 observations. The verification is left to you.

10.1.4 Leakage

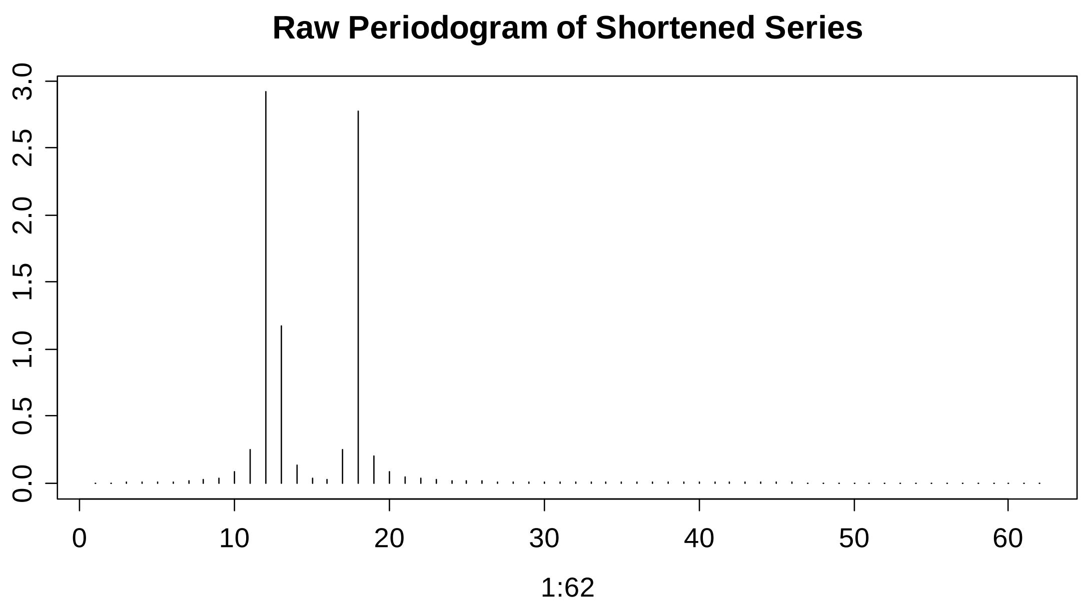
While some basic inspections of the periodogram can and sometimes do already provide valuable insight, there are a few issues which need to be taken care of. The first one which is discussed here is the phenomenon called *leakage*. It appears if there is no Fourier frequency that corresponds to the true periodicity in the data. Usually, the periodogram then shows higher values in the vicinity of the true frequency. The following simulation example is enlightening:

$$X_t = \cos\left(\frac{2\pi \cdot 13 \cdot t}{140}\right) + 0.8 \cdot \cos\left(\frac{2\pi \cdot 20 \cdot t}{140}\right), \text{ for } t = 0, ..., 139$$

We have a series of 140 observations which is made up as the superposition of two harmonic oscillations with frequencies $13/140$ and $20/140$. These correspond to periods of 7.00 and 10.77, and both are Fourier frequencies. We display the time series plot, as well as the periodogram:



Now if we shorten this very same series by 16 data points so that 124 observations remain, the true frequencies $20/140$ and $13/140$ do no longer appear in the decomposition model, i.e. are not Fourier frequencies anymore. The periodogram now shows leakage:



If not all of the true frequencies in the data generating model are Fourier frequencies, then, $\hat{\alpha}_k, \hat{\beta}_k$ from the decomposition model are only approximations to the true contribution of a particular frequency for the variation in the series.

11 State Space Models

State space modeling is a very flexible tool that can be applied in almost all applied fields. It would certainly merit a full course focusing on applied aspects on its own. Due to restrictions in time, we here provide only a small overview on the potential of the method. While it is possible to write most time series models in state space formulation, it is usually simpler to do without and use their own genuine notation. The real benefits of state space models only become unveiled when one has to deal with observations that are blurred with additional noise, or with situations, where some parameters are required to adapt over time.

We will here first introduce the general formulation of state space models, and then illustrate with a number of examples. The first two concern AR processes with additional observation noise, and the latter two are dynamic linear models, i.e. regression problems with time-varying coefficients, and the growth model.

The coefficients of state space models are usually estimated with the Kalman Filter. Because this is mathematically rather complex and in the primary focus of the user, this scriptum does not provide many details about it.

11.1 State Space Formulation

State space models are built on two equations. One is the state equation, and the other is the observation equation. We here introduce the general notation; their meaning will become clearer with examples discussed below.

State Equation

The values of the state at time t are represented by a column matrix X_t, and are a linear combination of the values of the state at time $t-1$ and random variation (system noise) from a multivariate normal distribution. The linear combination of values of the state at time $t-1$ is defined with a matrix G_t, and the covariance matrix of the multivariate normal is denoted with w_t.

$$X_t = G_t X_{t-1} + W_t \text{, where } W_t \sim N(0, w_t)$$

Observation Equation

The observation at time t is denoted by a column matrix Y_t that is a linear combination of the states, determined by a matrix F_t, and random variation (measurement noise) from a normal distribution with covariance matrix v_t.

$$Y_t = F_t X_t + V_t \text{, where } V_t \sim N(0, v_t)$$

Note that in this general formulation, all matrices can be time varying, but in most of our examples, they will be constant. Also, the nomenclature is different depending on the reference, but we here adopt the notation of **R**.

11.2 AR Processes with Measurement Noise

We are interested in a stochastic process X_t (which may be an AR process). Then, one usually makes measurements to obtain observations, i.e. acquires a realization of the process. So far, we operated under the assumption that the measurements were error-free, i.e. that there was no measurement noise. In many cases, this is hardly realistic, and we may rather have realizations of some random variable

$$Y_t = X_t + V_t \text{, where } V_t \sim N(0, \sigma_V^2) .$$

Thus, the realizations of the process of interest, X_t are latent, i.e. hidden under some random noise. We will now discuss how this issue can be solved in practice.

Example: AR(1)

As the simplest example of a state space model, we consider an AR(1) process which is superposed with some additional measurement noise. The state equation is as follows:

$$X_t = \alpha_1 X_{t-1} + W_t .$$

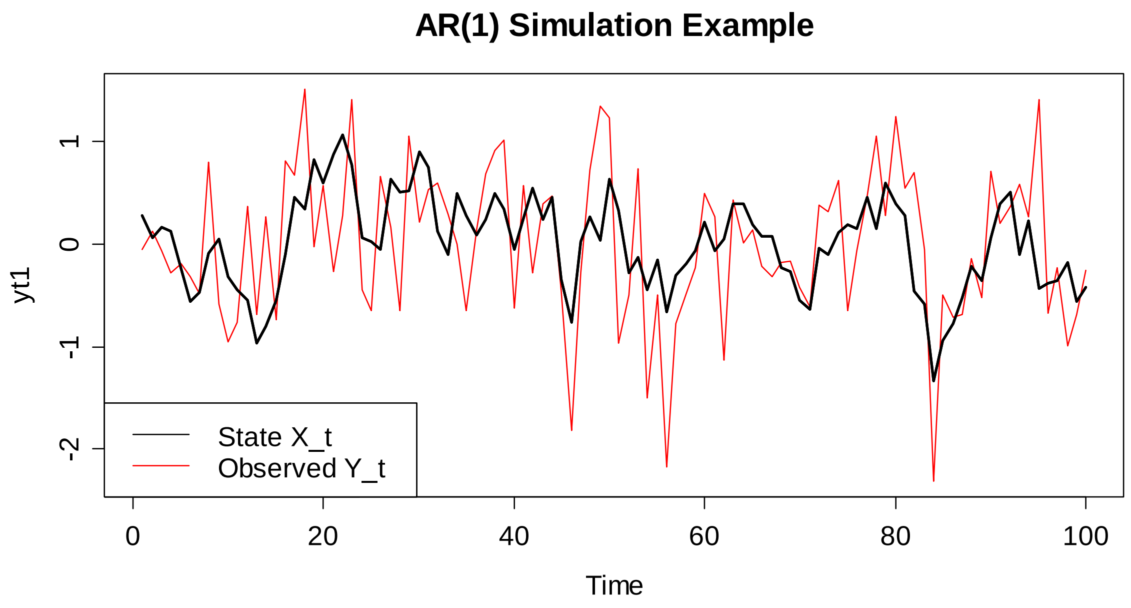
We assume that W_t is an iid innovation with Gaussian distribution, i.e. $W_t \sim N(0, \sigma_W^2)$. Also note that matrix G_t has dimension 1×1, is time-constant and equal to α_1. Under some additional measurement noise, our observations can be perceived as realizations of the random variable Y_t:

$$Y_t = X_t + V_t \text{, where } V_t \sim N(0,\ \sigma_V^2) .$$

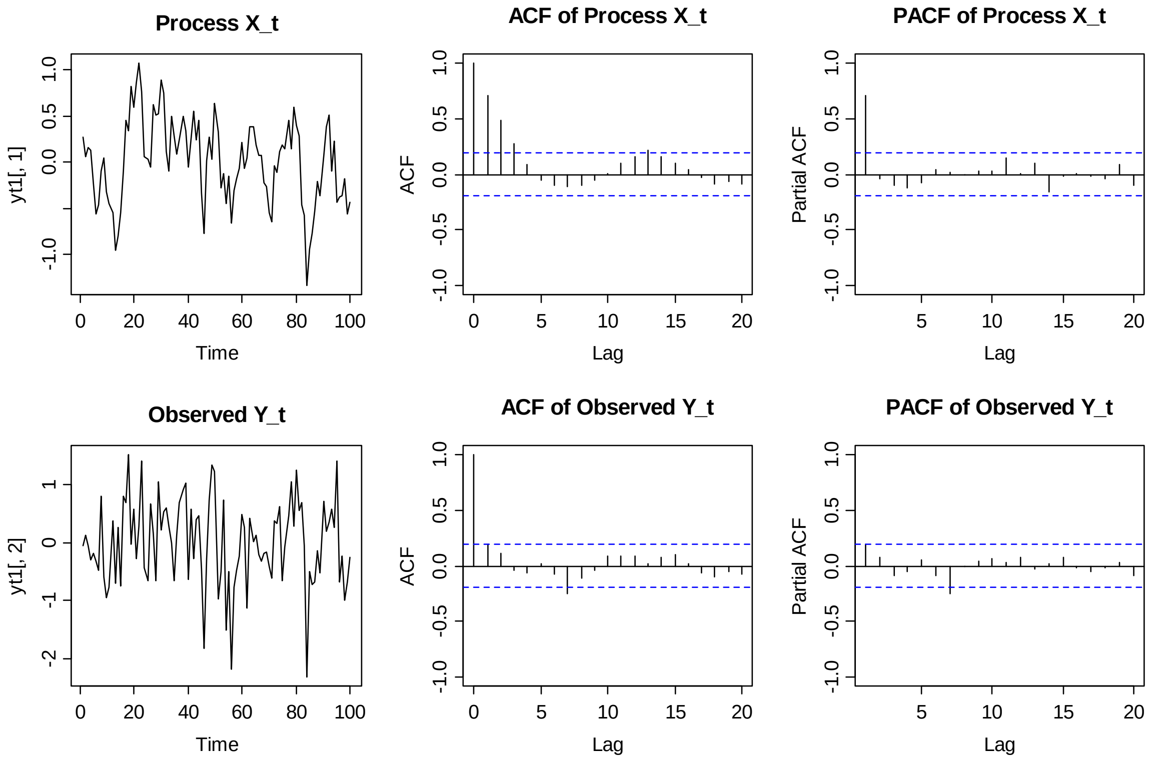
This is the observation equation, note that F_t is also time-constant and equal to the 1×1 identity matrix. We here assume that the errors V_t are iid, and also independent of X_s and W_s for all t and s. It is important to note that W_t is the process innovation that impacts future instances X_{t+k}. In contrast, V_t is pure measurement noise with no influence on the future of process X_t.

For illustration, we consider a simulation example. We use $\alpha_1 = 0.7$, the innovation variance σ_W^2 is 0.1 and the measurement error variance σ_V^2 is 0.5. The length of the simulated series is 100 observations. On the next page, we show a number of plots. They include a time series plot with both series X_t and Y_t, and the individual plots of X_t with its ACF/PACF, and of Y_t with ACF/PACF.

We clearly observe that the appearance of the two processes is very different. While X_t looks like an autoregressive process, and has ACF and PACF showing the stylized facts very prominently, Y_t almost appears to be White Noise. Please note that this is not true. There is some dependency also in Y_t, but it is blurred by some very strong noise component.

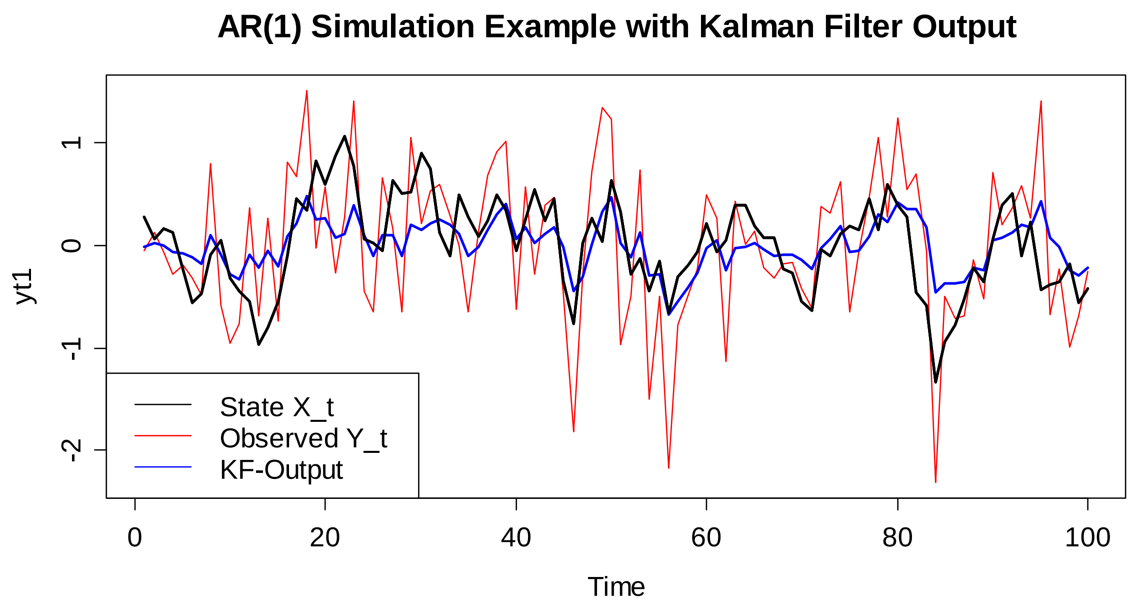


We here emphasize, that the state space formulation allowed to write a model comprised of a true signal plus additional noise. However, if we face an observed series of this type, we are not any further yet. We need some means to separate the two components. Kalman filtering allows doing so. In **R** package `sspir`, there are procedures that do the job, but they require a correctly formulated state space model as an input. The next page shows the details.



```
## Load the package for Kalman filtering
library(sspir)

## State Space Formulation
ssf <- SS(y = as.matrix(obs),
          Fmat = function(tt,x,phi) { return(matrix(1)) },
          Gmat = function(tt,x,phi) { return(matrix(0.7)) },
          Vmat = function(tt,x,phi) { return(matrix(0.5)) },
          Wmat = function(tt,x,phi) { return(matrix(0.1)) },
          m0 = matrix(0),
          C0 = matrix(0.1))

## Kalman Filtering
fit  <- kfilter(ssf)
plot(fit$m, col="blue", lwd=2, ...)
```

Kalman filter in R requires to specifiy the state space model first. We need to supply argument **y** which stands for the observed time series data. They have to come in the form of a matrix. Moreover, we have to specify the matrices F_t, G_t, as well as the covariance structures v_t, w_t. In our case, these are all simple 1×1 matrices. Finally, we have to provide `m0`, the starting value of the initial state, and `C0`, the variance of the initial state.



We can then employ the Kalman filter to recover the original signal X_t. It was added as the blue line in the above plot. While it is not 100% accurate, it still does a very good job of filtering the noise out. However, note that with this simulation example, we have some advantage over the real simulation. It was easy for us to specify the correct state space formulation. In practice, we might have problems to identify good values for G_t (the true AR(1) parameter) and the variances in v_t, w_t. On the other hand, in practice the precision of many measurement devices is more or less known, and thus some educated guess is possible.

Example: AR(2)

Here, we demonstrate the formulation of a state space model for an AR(2) process that is superimposed with some measurement error. This example is important, because now, we need to use matrices in the state equation. It is as follows:

$$\begin{pmatrix} X_t \\ X_{t-1} \end{pmatrix} = \begin{pmatrix} \alpha_1 & \alpha_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} X_{t-1} \\ X_{t-2} \end{pmatrix} + \begin{pmatrix} W_t \\ 0 \end{pmatrix}$$

Apparently, this is a two-dimensional model. The observation equation is:

$$Y_t = (1 \quad 0) \cdot \begin{pmatrix} X_t \\ X_{t-1} \end{pmatrix} + V_t$$

Once the equations are set up, it is straightforward to derive the matrices:

$$G_t = G = \begin{pmatrix} \alpha_1 & \alpha_2 \\ 0 & 0 \end{pmatrix}, \; H_t = H = (1 \quad 0), \; w_t = \begin{pmatrix} \sigma_W^2 & 0 \\ 0 & 0 \end{pmatrix}, \; v_t = \sigma_V^2$$

Similar to the example above, we could now simulate from an AR(2) process, add some artificial measurement noise and then try to uncover the signal using the Kalman filter. This is left as an exercise.

11.3 Dynamic Linear Models

A specific, but very useful application of state space models is to generalize linear regression such that the coefficients can vary over time. We consider a very simple example where the sales manager in a house building company uses the following model: the company's house sales at time t, denoted as S_t, depends on the general levels of sales in that area L_t and the company's pricing policy P_t.

$$S_t = L_t + \beta_t P_t + V_t$$

This is a linear regression model with price as the predictor, and the general level as the intercept. The assumption is that their influence varies over time, but generally only in small increments. We can use the following notation:

$$L_t = L_{t-1} + \Delta L_t$$

$$\beta_t = \beta_{t-1} + \Delta \beta_t$$

In this model, we assume that v_t, ΔL_t and $\Delta \beta_t$ are random deviations with mean zero that are independent over time. While we assume independence of ΔL_t and $\Delta \beta_t$, we could also allow for correlation among the two. The relative magnitudes of these perturbations are accounted for with the variances in the matrices V_t and W_t of the state space formulation. Note that if we set $W_t = 0$, then we are in the case of plain OLS regression with constant parameters. Hence, we can also formulate any regression models in state space form. Here, we have:

$$Y_t = S_t,\ X_t = \begin{pmatrix} L_t \\ \beta_t \end{pmatrix},\ W_t = \begin{pmatrix} \Delta L_t \\ \Delta\beta_t \end{pmatrix},\ F_t = \frac{1}{P_t},\ G = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

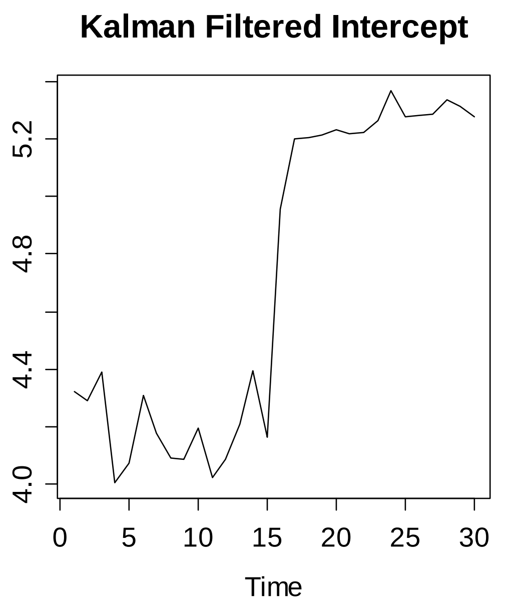
Because we do not have any data for this sales example, we again rely on a simulation. Apparently, this also features the advantage that we can evaluate the Kalman filter output versus the truth. Thus, we let

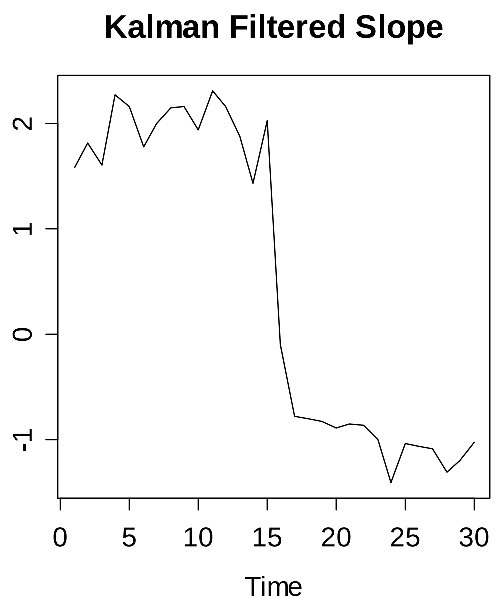
$$y_t = a + bx_t + z_t$$

$$x_t = 2 + t/10$$

We simulate 30 data points from $t = 1,...,30$ and assume errors which are standard normally distributed, i.e. $z_t \sim N(0,1)$. The regression coefficients are $a = 4$ and $b = 2$ for $t = 1,...,15$ and $a = 5$ and $b = -1$ for $t = 16,...,30$. We will fit a straight line with time-varying coefficients, as this is the model that matches what we had found for the sales example above.

```
## Simulation
set.seed(1)
x1    <- 1:30
x1    <- x1/10+2
aa    <- c(rep(4,15), rep( 5,15))
bb    <- c(rep(2,15), rep(-1,15))
nn    <- length(x1)
y1    <- aa+bb*x1+rnorm(nn)
x0    <- rep(1,nn)
xx    <- cbind(x0,x1)
x.mat <- matrix(xx, nrow=nn, ncol=2)
y.mat <- matrix(y1, nrow=nn, ncol=1)

## State Space Formulation
ssf <- SS(y=y.mat, x=x.mat,
          Fmat=function(tt,x,phi)
            return(matrix(c(x[tt,1],x[tt,2]),2,1)),
          Gmat=function(tt,x,phi) return(diag(2)),
          Wmat=function(tt,x,phi) return(0.1*diag(2)),
          Vmat=function(tt,x,phi) return(matrix(1)),
          m0=matrix(c(5,3),1,2),
          C0=10*diag(2))

## Kalman-Filtering
fit <- kfilter(ssf)
par(mfrow=c(1,2))
plot(fit$m[,1], type="l", xlab="Time", ylab="")
title("Kalman Filtered Intercept")
plot(fit$m[,2], type="l", xlab="Time", ylab="")
title("Kalman Filtered Slope")
```





The plots show the Kalman filter output for intercept and slope. The estimates pick up the true values very quickly, even after the change in the regime. It is worth noting that in this example, we had a very clear signal with relatively little noise, and we favored recovering the truth by specifying the state space formulation with the true error variances that are generally unknown in practice.