timeseRies

Léo Belzile

version of 2019-02-26

Contents

Preliminary remarks

This is a web complement to MATH 342 (Time Series), a third year bachelor course offered at EPFL.

We shall use the **R** programming language througout the course (as it is free and it is used in other statistics courses at EPFL). Visit the R-project website[1] to download the program. The most popular graphical cross-platform front-end is RStudio Desktop[2].

R is an object-oriented interpreted language. It differs from usual programming languages in that it is designed for interactive analyses.

Since **R** is not a conventional programming language, my teaching approach will be learning-by-doing. The benefit of using *Rmarkdown* is that you see the output directly and you can also copy the code.

[1] https://cran.r-project.org/
[2] https://www.rstudio.com/products/rstudio/download/

Chapter 1

Introduction

This online tutorial for MATH 342 is meant to provide a review of basic R syntax, including plotting functions.

You should install **R** from https://cran.r-project.org/. I highly advise that you also install the **RStudio** IDE[1] to facilitate your analysis. If you have never touched **R**, find a tutorial online to grasp the basics of the programming language, for example Wickham's R for Data Science book[2]. Many websites provide overview of exploratory data analysis (EDA)[3].

R is a programming language that compiles in real-time, meaning that you can simply type instructions in the console to see them executed.

If you have not used **R** before, work through some of the introduction at http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf. If you have used **R** before, work through some of David S. Stoffer's examples[4] to get an idea of some of the time series functions. We will use them more systematically in future weeks.

I will heavily focus on some time series libraries that are not part of the `base` or `stat`. The latter are the default libraries that are installed alongside with the base R. They contain `ts`, `acf`, etc. These standard tools are very useful, except that they do not handle irregular time series or missing values. These functions will likely not be altered (or improved) in the future for reproducibility reasons. Many other contributed **R** libraries are more intuitive and easy to use than the base functions. However, living on the cutting edge means that functions may change or stop working anytime in the future.

The first thing to know about R is how to access help files. If you want to read about time series, type `help.search("time series")`. If you want to read the help file on a particular function, for example `plot`, use `?plot` or `help("plot")`.

1.1 Exploratory Data Analysis

1.1.1 Libraries

We will use many functions from the `base` package, which is loaded by default, but also some functions from time series libraries. What makes the R programming language so great is its vast contributed packages libraries. An exhaustive list of these can be found at https://cran.r-project.org/web/views/TimeSeries.html. Let's install some of these.

[1] http://rstudio.com
[2] http://r4ds.had.co.nz/
[3] http://r4ds.had.co.nz/exploratory-data-analysis.html
[4] http://www.stat.pitt.edu/stoffer/tsa4/

```
# Most common time series libraries
install.packages(c("dlm", "forecast", "lubridate", "tsa", "tseries", "xts",
    "zoo"), dependencies = TRUE)
# Datasets
install.packages(c("expsmooth", "fpp", "TSA", "astsa"))
# Hadley Wickham's tidyverse universe
install.packages("tidyverse")

# To load a library, use the function
library(xts)
```

As a side remark, note that the `tidyverse`, which loads a bundle of packages, overwrites some of the base functions, notably `lag` and `filter` which are present in both `dplyr` and `stats` (one of the default libraries that come alongside with **R** and is loaded upon start). Load libraries with great caution! In case of ambiguity (when many functions in different packages have the same name), use the `::` operator to specify the package, e.g. `stat::lag`. You can unload a library using the command

```
detach("package:tidyverse", unload = TRUE)
```

1.1.2 Loading datasets

You can load and read objects, whether `txt`, `csv` from your computer or by directly downloading them into R from the web. You can call R datasets found in packages via `data()`

Good data sources for your semester projects are

- Rob Hyndman's Time Series Data Library[5]
- Mike West's datasets[6]
- Ruey Tsay's datasets from the book *Analysis of Financial Time Series*[7]
- Bo Li's paleoclimate datasets[8]
- Kaggle datasets[9]
- Don Percival's data page (navigate to Data tab)[10]
- Carbon dioxide data[11]
- Climate Research Unit[12]
- NOAA[13]

1.1.3 Time series objects and basic plots

Objects in **R** are vectors by default, which have a type and attributes (vector is a type, `length` is an attribute of vectors). Some objects also inherit a class, such as `ts`. They inherit printing and plotting methods specific to the data class.

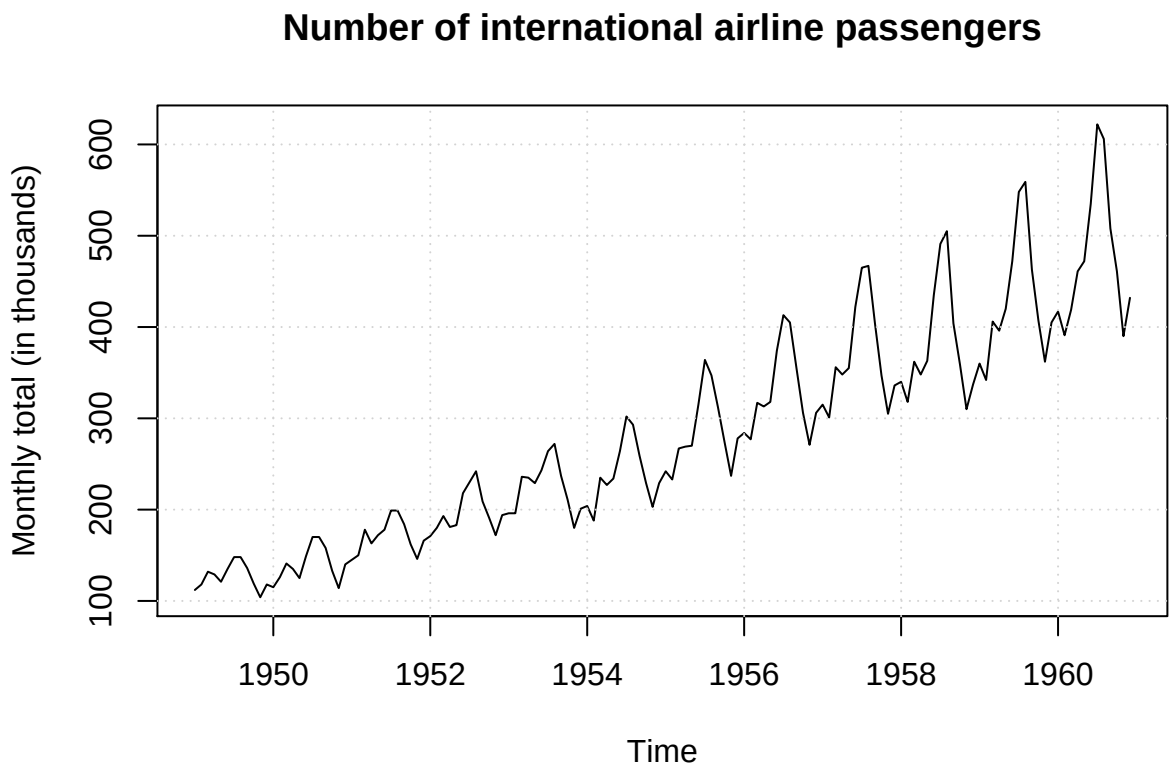
[5] https://datamarket.com/data/list/?q=provider%3Atsdl
[6] http://www2.stat.duke.edu/~mw/ts_data_sets.html
[7] https://faculty.chicagobooth.edu/ruey.tsay/teaching/fts/
[8] http://publish.illinois.edu/boli-uiuc/paleoclimate/
[9] https://www.kaggle.com/datasets
[10] http://faculty.washington.edu/dbp/s519/
[11] http://cdiac.esd.ornl.gov/trends/co2/contents.htm
[12] http://www.cru.uea.ac.uk/
[13] https://data.noaa.gov/dataset

We start by loading the `AirPassengers` dataset, which contains monthly airline passenger numbers for years 1949-1960. Datasets that are found in libraries other than `datasets` must typically be loaded via a call to `data`, unless they are lazy loaded when calling the library. Both datasets are time series.

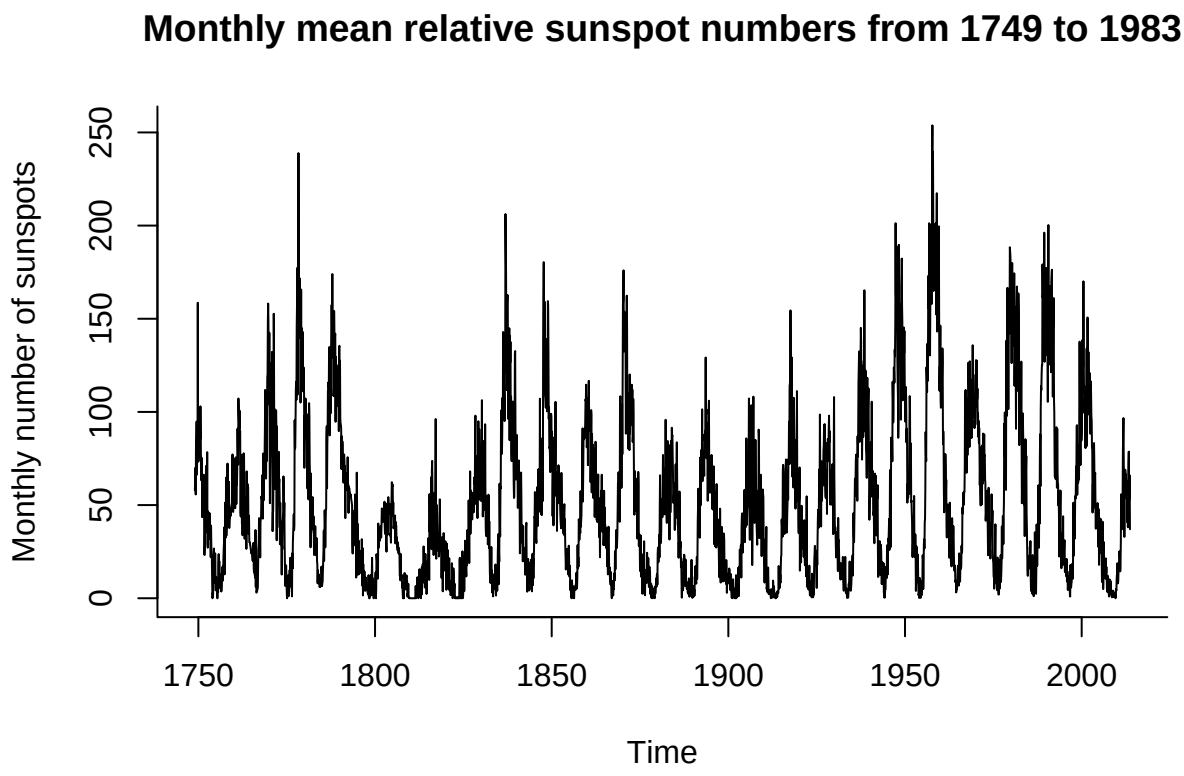
```
# AirPassenger dataset, lazy-loaded
class(AirPassengers)  #object of class 'ts'
```

```
[1] "ts"
```

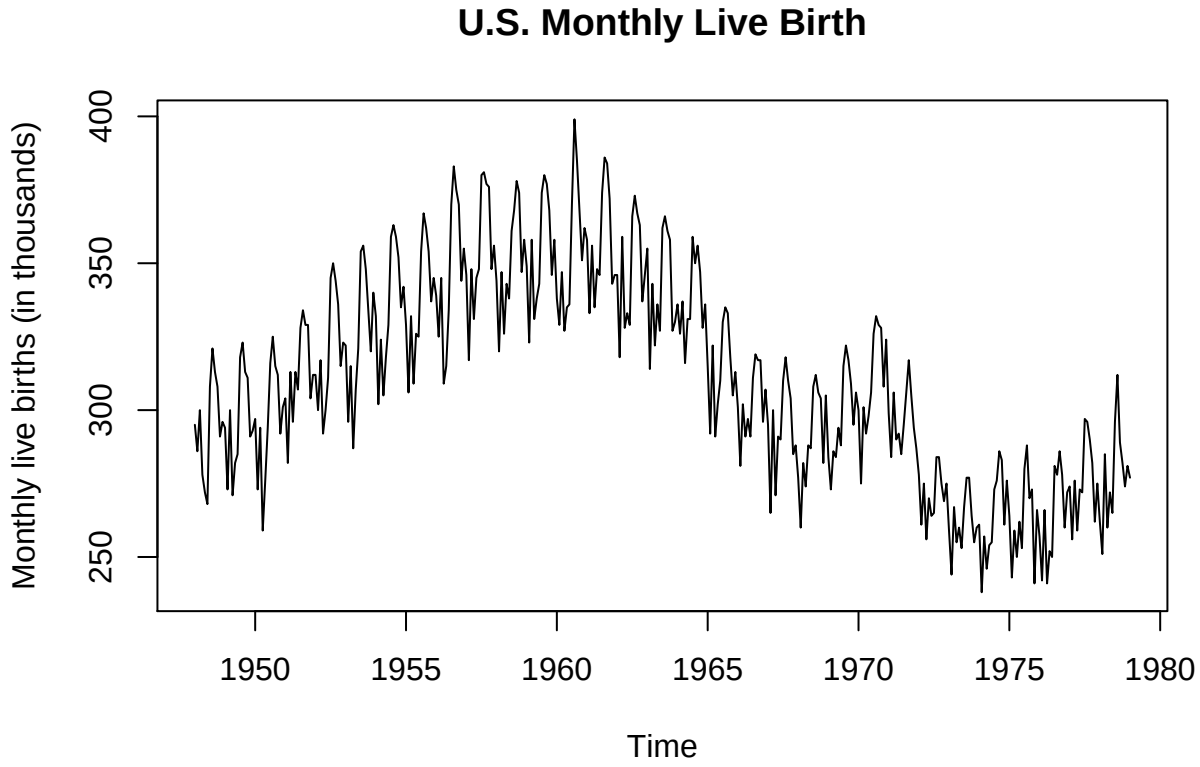
```
`?`(AirPassengers  #description of the dataset
)
# Basic plot
plot(AirPassengers, ylab = "Monthly total (in thousands)", main = "Number of international airline passe
grid()
```




```
`?`(sunspot.month)
plot(sunspot.month, ylab = "Monthly number of sunspots", main = "Monthly mean relative sunspot numbers
    bty = "l")
```




```r
# Dataset present in a R package - without loading the package
data(list = "birth", package = "astsa")
plot(birth, ylab = "Monthly live births (in thousands)", main = "U.S. Monthly Live Birth")
```



1.2 Introduction to the basic time series functions

The first example we are going to handle is `lh`, a time series of 48 observations at 10-minute intervals on luteinizing hormone levels for a human female. Start by printing it.

```
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 3.2 3.2 2.7
[18] 2.2 2.2 1.9 1.9 1.8 2.7 3.0 2.3 2.0 2.0 2.9 2.9 2.7 2.7 2.3 2.6 2.4
[35] 1.8 1.7 1.5 1.4 2.1 3.3 3.5 3.5 3.1 2.6 2.1 3.4 3.0 2.9
```

Look at the information: `Start = 1`, `End = 48` and `Frequency = 1`.

The second example, `deaths`, gives monthly deaths in the UK from a set of common lung diseases for the years 1974 to 1979.

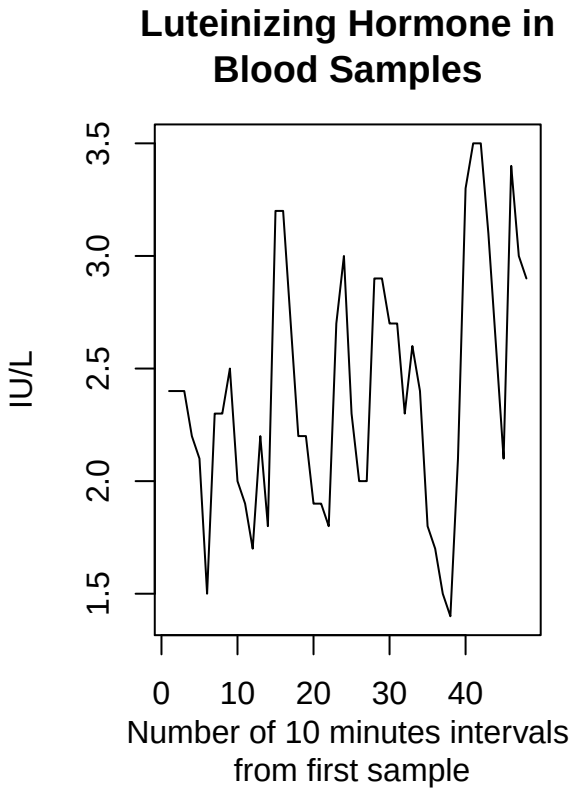
```
data("deaths", package = "MASS")
deaths
```

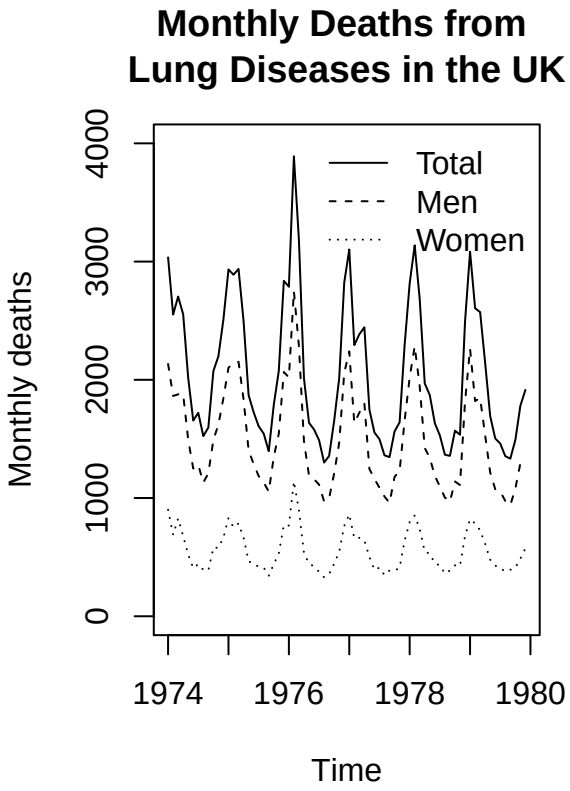
```
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1974 3035 2552 2704 2554 2014 1655 1721 1524 1596 2074 2199 2512
1975 2933 2889 2938 2497 1870 1726 1607 1545 1396 1787 2076 2837
1976 2787 3891 3179 2011 1636 1580 1489 1300 1356 1653 2013 2823
1977 3102 2294 2385 2444 1748 1554 1498 1361 1346 1564 1640 2293
1978 2815 3137 2679 1969 1870 1633 1529 1366 1357 1570 1535 2491
1979 3084 2605 2573 2143 1693 1504 1461 1354 1333 1492 1781 1915
```

Use `tsp(deaths)` to get `Start = 1974`, `End = 1979.917` and `Frequency = 12`. You can also access each of these attributes using the functions `start(deaths)`, `end(deaths)` and `frequency(deaths)`. Use `cycle(deaths)` to get the position in the cycle of each observation.

Time series can be plotted by `plot`. The argument `lty` of the function `plot` controls the type of the plotted line (solid, dashed, dotted, ...). For more details, type `?par` (for graphical parameters).

```
par(mfrow = c(1, 2))  #2 plot side by side
plot(lh, main = "Luteinizing Hormone in\nBlood Samples", ylab = "IU/L", xlab = "Number of 10 minutes in
plot(deaths, main = "Monthly Deaths from \nLung Diseases in the UK", ylab = "Monthly deaths",
    ylim = c(0, 4000))
lines(mdeaths, lty = 2)
lines(fdeaths, lty = 3)
legend(x = "topright", bty = "n", legend = c("Total", "Men", "Women"), lty = c(1,
    2, 3))
```






```
graphics.off()  #close console
```

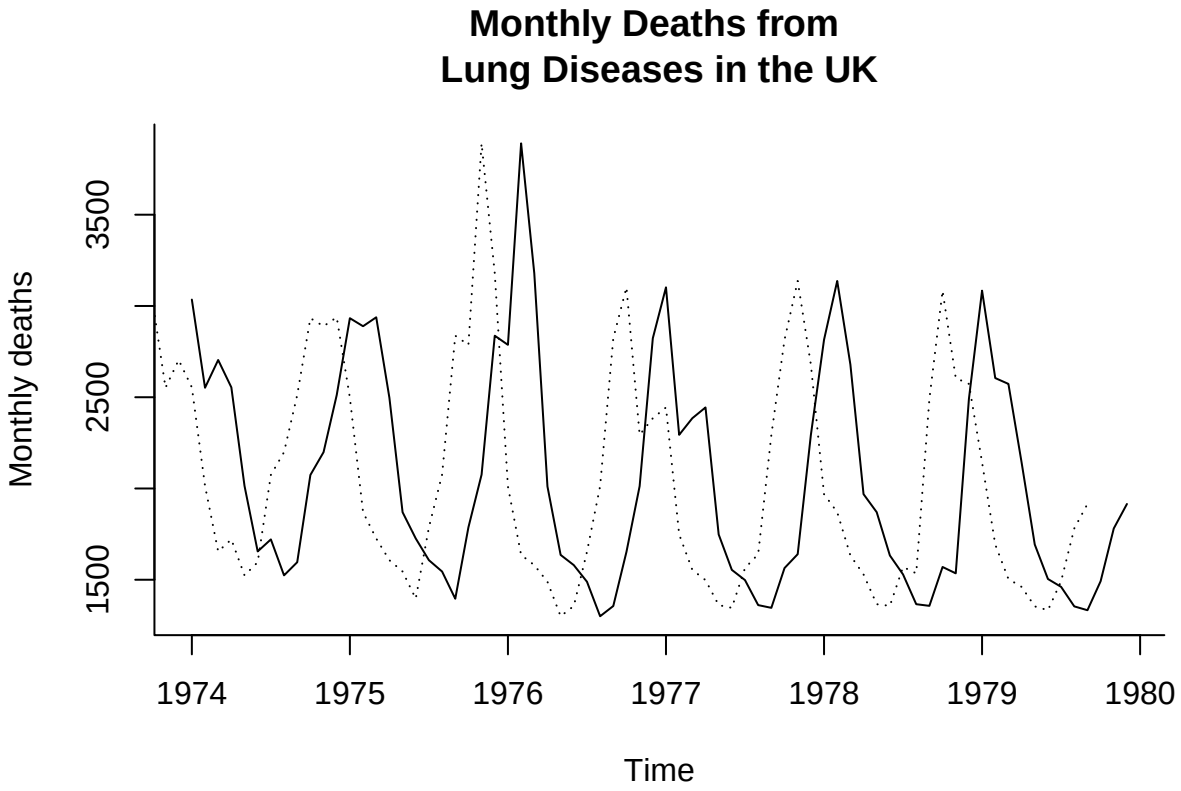
Above, you can see plots of `lh` and the three series on deaths. In the right-hand plot, the dashed series is for males, the dotted series for females and the solid line for the total.

The functions `ts.union` and `ts.intersect` bind together multiple time series which have a common frequency. The time axes are aligned and only observations at times that appear in all the series are retained with `ts.intersect`; with `ts.union` the combined series covers the whole range of the components, possibly as `NA` values.

The function `window` extracts a sub-series of a single or multiple time series, by specifying `start`, `end` or both.

The function `lag` shifts the time axis of a series back by k positions (default is `k = 1`). Thus `lag(deaths, k=3)` is the series of deaths shifted one quarter into the past.

```
plot(deaths, main = "Monthly Deaths from \nLung Diseases in the UK", ylab = "Monthly deaths",
    bty = "l")
lines(lag(deaths, k = 3), lty = 3)
```



The function `diff` takes the difference between a series and its lagged values and so returns a series of length $n-k$ with values lost from the beginning (if $k>0$) or end. Beware: the argument `lag` (default is `lag = 1`) is used in the usual sense here, so `diff(deaths, lag=3)` is equal to `deaths - lag(deaths, k=-3)`! The function `diff` has an argument `differences` which causes the operation to be iterated.

The function `aggregate` can be used to change the frequency of the time base.

```
aggregate(deaths, 4, sum)
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
1974 8291 6223 4841 6785
1975 8760 6093 4548 6700
1976 9857 5227 4145 6489
1977 7781 5746 4205 5497
1978 8631 5472 4252 5596
1979 8262 5340 4148 5188
```

```
aggregate(deaths, 1, sum)
```

```
Time Series:
Start = 1974
End = 1979
Frequency = 1
[1] 26140 26101 25718 23229 23951 22938
```

```
aggregate(deaths, 4, mean)
```

```
         Qtr1     Qtr2     Qtr3     Qtr4
```

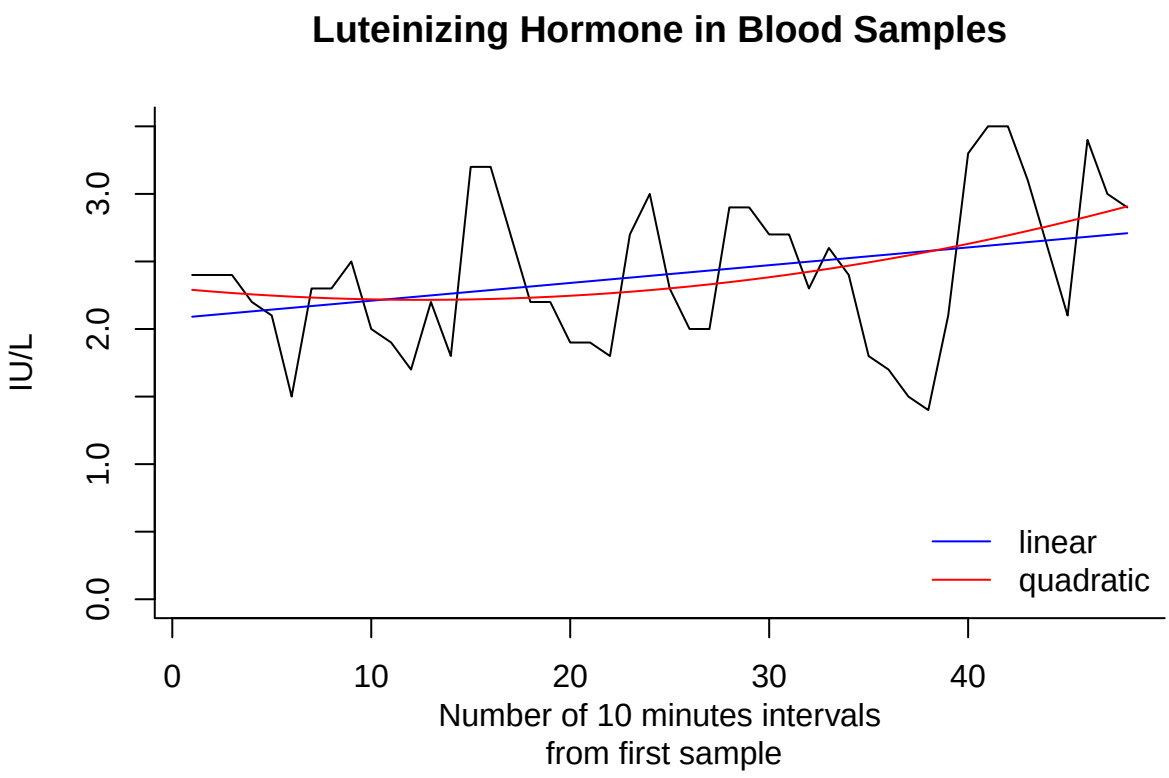
```
1974 2763.667 2074.333 1613.667 2261.667
1975 2920.000 2031.000 1516.000 2233.333
1976 3285.667 1742.333 1381.667 2163.000
1977 2593.667 1915.333 1401.667 1832.333
1978 2877.000 1824.000 1417.333 1865.333
1979 2754.000 1780.000 1382.667 1729.333
```

```
aggregate(deaths, 1, mean)
```

```
Time Series:
Start = 1974
End = 1979
Frequency = 1
[1] 2178.333 2175.083 2143.167 1935.750 1995.917 1911.500
```

One way to compute the linear or polynomial trend of a series is to use the function `lm`, which fits linear models. The function `fitted` allows you to extract the model fitted values, while `c(1:48)` represents the integers from 1 to 48 and the function `poly` computes orthogonal polynomials.

```
plot(lh, main = "Luteinizing Hormone in Blood Samples", ylab = "IU/L", xlab = "Number of 10 minutes int
    bty = "l", ylim = c(0, 3.5))
lines(fitted(lm(lh ~ c(1:48))), col = "blue")
lines(fitted(lm(lh ~ poly(1:48, 2))), col = "red")
legend(x = "bottomright", legend = c("linear", "quadratic"), col = c(4, 2),
    lty = c(1, 1), bty = "n")
```



1.2.1 Exercise 1: Beaver temperature

1. Load the `beav2` data from the library `MASS`.
2. Examine the data frame using `summary`, `head`, `tail`. Query the help with `?beav2` for a description of the dataset
3. Transform the temperature data into a time series object and plot the latter.
4. Fit a linear model using `lm` and the variable `activ` as factor, viz. `lin_mod <- lm(temp~as.factor(activ), data=beav2)`. Overlay the means on your plot with `lines(fitted(lin_mod))` replacing `lin_mod` with your `lm` result.
5. Inspect the residuals (`resid(lin_mod)`) and determine whether there is any evidence of trend or seasonality.
6. Look at a quantile-quantile (Q-Q) plot to assess normality. You can use the command `qqnorm` if you don't want to transform manually the residuals with `qqline` or use `plot(lin_mod, which=2)`.
7. Plot the lag-one residuals at time t and $t-1$. Is the dependence approximately linear?

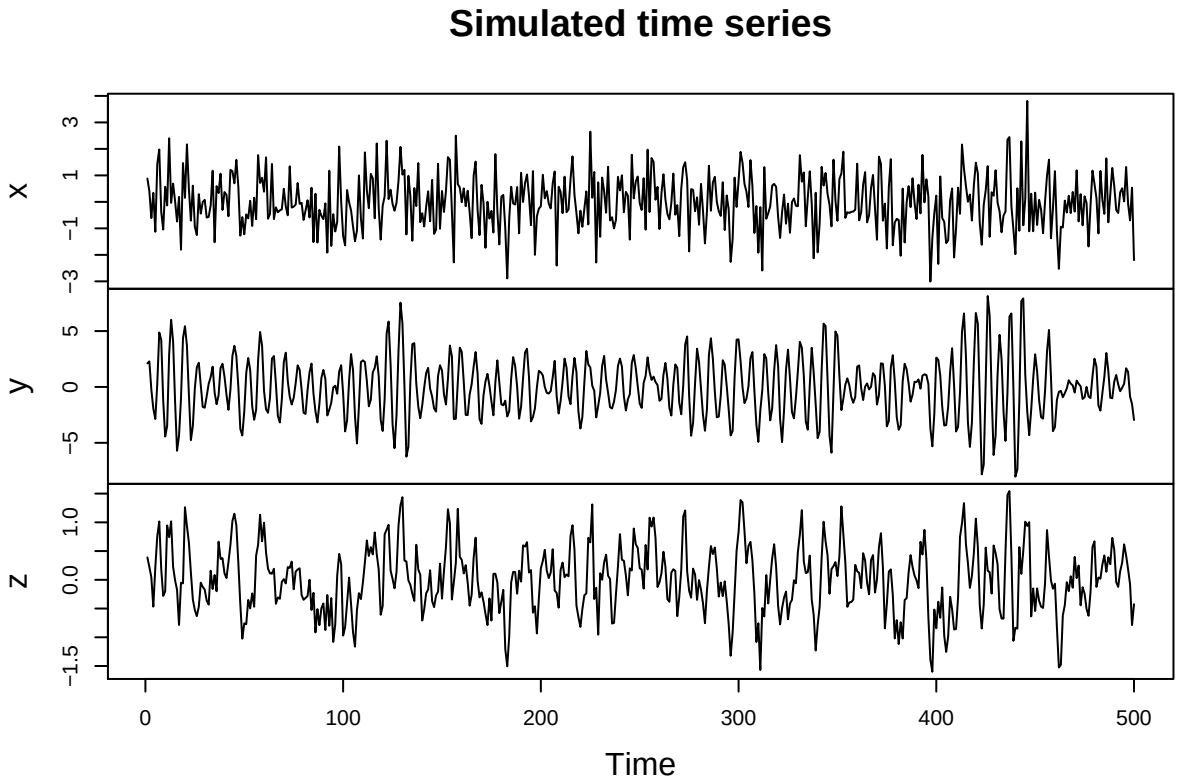
1.3 Second order stationarity

The example below corresponds to examples 1.9 and 1.10 from Shumway and Stoffer. It shows how to create MA and AR series based on white noise using the `filter` function. It is best practice when simulating autoregressive models to burn-in (discard) the first few iterations to remove the dependencies on the starting values (here zeros). Note that the function `filter` returns a `ts` object.

```
set.seed(1)
x <- rnorm(550, 0, 1)  # Samples from N(0,1) variates
y <- filter(x, method = "recursive", filter = c(1, -0.9))  #autoregression
z <- filter(x, method = "convolution", filter = rep(1/3, 3), sides = 2)  # moving average
class(z)
```

```
[1] "ts"
```

```
x <- x[-c(1:49, 550)]
y <- y[-c(1:49, 550)]
z <- z[-c(1:49, 550)]
# ts.union if we did not remove values (and the class `ts`)
plot.ts(cbind(x, y, z), main = "Simulated time series")
```



We notice immediately that the MA process looks smoother than the white noise, and that the innovations (peaks) of the AR process are longer lasting. If we had not simulated more values and kept some, the first and last observations of z would be `NA`s. The series created are of the form

$$z_t = \frac{1}{3}(x_{t-1} + x_t + x_{t+1})$$

and

$$y_t = y_{t-1} - 0.9y_{t-2} + x_t.$$

The correlogram provides an easy to use summary for detecting *linear* dependencies based on correlations. The function `acf` will return a plot of the correlogram, by default using the correlation. Unfortunately, the basic graph starts at lag 1, which by default has correlation 1 and thus compress the whole plot unnecessarily. Blue dashed lines indicate 5% critical values at $\pm 1.96/\sqrt{n}$ under the null hypothesis of white noise (*not* stationarity).

The function `acf` computes and plots c_t and r_t, the estimators for the autocovariance and autocorrelation functions

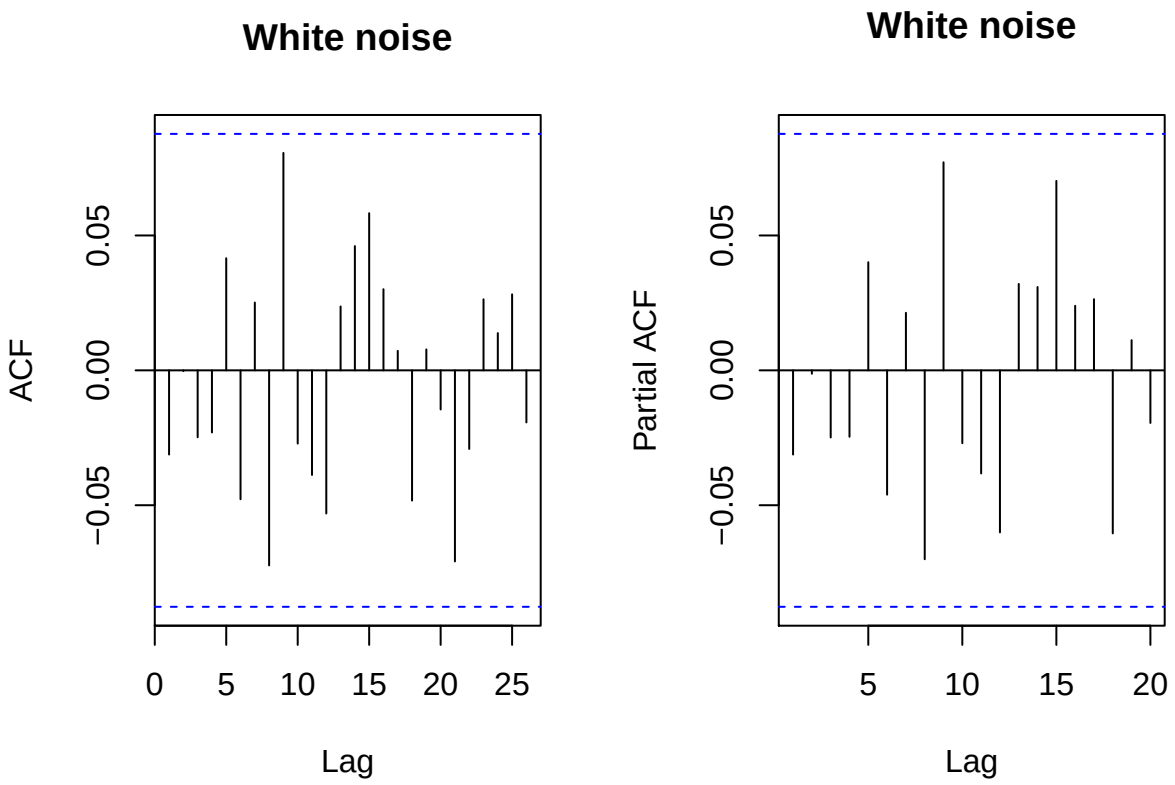
$$c_t = n^{-1} \sum_{s=\max(1,-t)}^{\min(n-t,n)} [X_{s+t} - \bar{X}][X_s - \bar{X}], \quad r_t = c_t / c_0.$$

The argument `type` controls which is used and defaults to the correlation. This is easily extended to several time series observed over the same interval

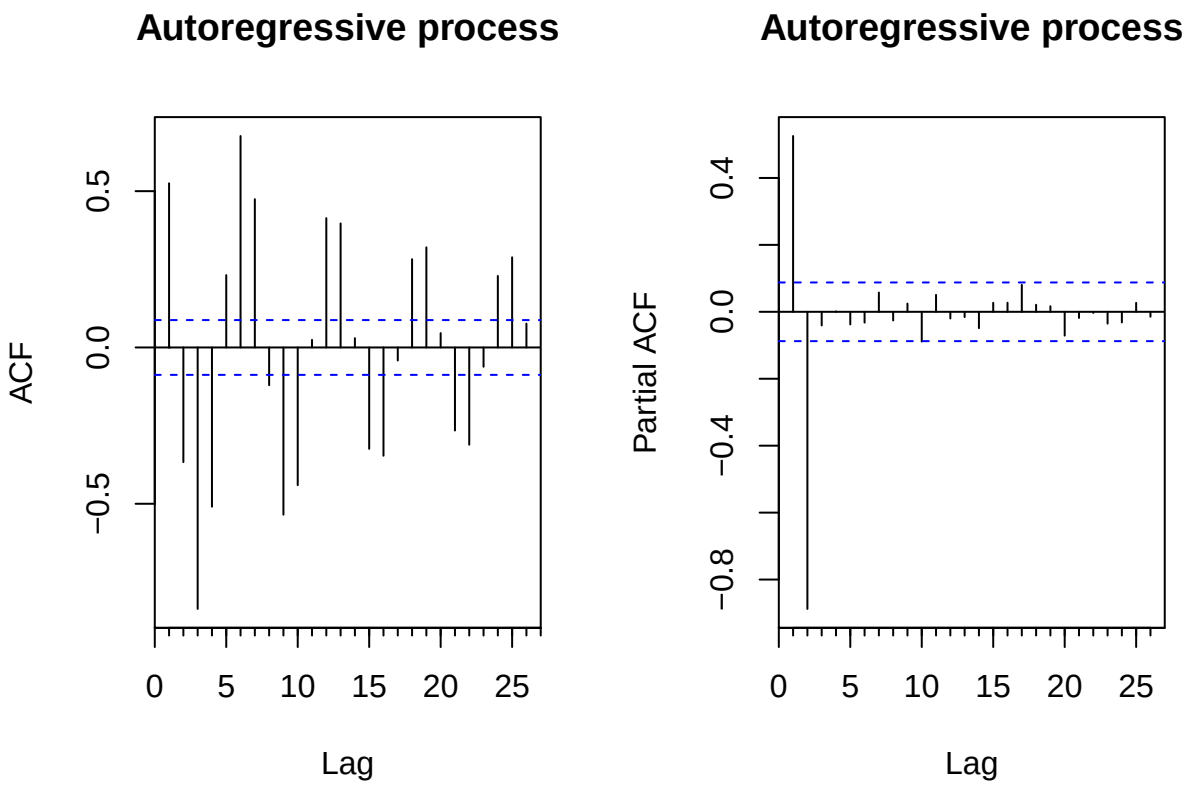
$$c_{ij}(t) = n^{-1} \sum_{s=\max(1,-t)}^{\min(n-t,n)} [X_i(s+t) - \bar{X}_i][X_j(s) - \bar{X}_j].$$

Unfortunately, the function `acf` always display the zero-lag autocorrelation, which is 1 by definition. This oftentimes squeezes the whole correlogram values and requires manual adjustment so one can properly view whether the sample autocorrelations are significant.

```r
# acf(x, lag.max=20, demean = TRUE, main = 'White noise') #WRONG
par(mfrow = c(1, 2))
TSA::acf(x, demean = TRUE, main = "White noise")
pacf(x, lag.max = 20, main = "White noise")
```




```r
# Equivalent functions from forecast
forecast::Acf(y, main = "Autoregressive process")
forecast::Acf(y, type = "partial", main = "Autoregressive process")  #or forecast::Pacf
```



You can thus use instead the function `forecast::Acf`, which removes the first lag of the autocorrelation function plot, or `TSA::acf`. The function `pacf` return the partial autocorrelations and the function `ccf` to compute the cross-correlation or cross-covariance of two univariate series

```
# Load datasets
data(deaths, package = "MASS")
data(lh, package = "datasets")
# Second order summaries
forecast::Acf(lh, main = "Correlogram of \nLuteinizing Hormone", ylab = "Autocorrelation")  #autocorrel
acf(lh, type = "covariance", main = "Autocovariance of\n Luteinizing Hormone")  #autocovariance
acf(deaths, main = "Correlogram of\n`deaths` dataset")
ccf(fdeaths, mdeaths, ylab = "cross-correlation", main = "Cross correlation of `deaths` \nfemale vs mal
# acf(ts.union(mdeaths,fdeaths)) # acf and ccf - multiple time series of
# male and female deaths
```

The plots of the `deaths` series shows the pattern of seasonal series, and the autocorrelations do not damp down for large lags. Note how one of the cross series is only plotted for negative lags. Plot in row 2 column 1 shows c_{12} for negative lags, a reflection of the plot of c_{21} for positive lags. For the cross-correlation, use e.g. `ccf(mdeaths, fdeaths, ylab="cross-correlation")`.

Plotting lagged residuals is a useful graphical diagnostic for detecting non-linear dependencies. The following function plots residuals at k different lags and may be useful for diagnostic purposes.

```
pairs.ts <- function(d, lag.max = 10) {
    old_par <- par(no.readonly = TRUE)
    n <- length(d)
    X <- matrix(NA, n - lag.max, lag.max)
    col.names <- paste("Time+", 1:lag.max)
    for (i in 1:lag.max) X[, i] <- d[i - 1 + 1:(n - lag.max)]
```

```
    par(mfrow = c(3, 3), pty = "s", mar = c(3, 4, 0.5, 0.5))
    lims <- range(X)
    for (i in 2:lag.max) plot(X[, 1], X[, i], panel.first = {
        abline(0, 1, col = "grey")
    }, xlab = "Time", ylab = col.names[i - 1], xlim = lims, ylim = lims, pch = 20,
        col = rgb(0, 0, 0, 0.25))
    par(old_par)
}
# Look at lag k residuals
pairs.ts(sunspots)
```

1.3.1 Exercise 2: SP500 daily returns

1. Download the dataset using the following command

```
sp500 <- tseries::get.hist.quote(instrument = "^GSPC", start = "2000-01-01",
    end = "2016-12-31", quote = "AdjClose", provider = "yahoo", origin = "1970-01-01",
    compression = "d", retclass = "zoo")
```

2. Obtain the daily percent return series and plot the latter against time.
3. With the help of graphs, discuss evidences of seasonality and nonstationarity. Are there seasons of returns?
4. Plot the (partial) correlogram of both the raw and the return series. Try the acf with `na.action=na.pass` and without (by e.g. converting the series to a vector using `as.vector`. Comment on the impact of ignoring time stamps.
5. Plot the (partial) correlogram of the absolute value of the return series and of the squared return series. What do you see?

1.4 Simulations

The workhorse for simulations from ARIMA models is `arima.sim`. To generate an AR(1) and an MA(1) processes using the function `arima.sim`, one can use.

```
ar1 <- arima.sim(n = 100, model = list(ar = 0.9))
ma1 <- arima.sim(n = 100, model = list(ma = 0.8))
```

Define the following function to generate an ARCH(1) process.

```
arch.sim1 <- function(n, a0 = 1, a1 = 0.9) {
    y <- eps <- rnorm(n)
    for (i in 2:n) y[i] <- eps[i] * sqrt(a0 + a1 * y[i - 1]^2)
    ts(y)
}
a0 <- 0.05
a1 <- 0.8
n <- 2000
y1 <- arch.sim1(n, a0, a1)
```

Use the second-order summaries functions to analyse the obtained process, the process of the squared values and the process of the absolute values.

Do it again with the following function, which has Student distributed variables as driving noise.

```
arch.sim2 <- function(n, df = 100, a0 = 1, a1 = 0.9) {
    y <- eps <- rt(n, df = df) * sqrt((df - 2)/df)
    for (i in 2:n) y[i] <- eps[i] * sqrt(a0 + a1 * y[i - 1]^2)
    ts(y)
}
a0 <- 0.05
a1 <- 0.8
n <- 2000
y1 <- arch.sim2(n, a0 = a0, a1 = a1)
```

1.4.1 Exercise 3: Simulated data

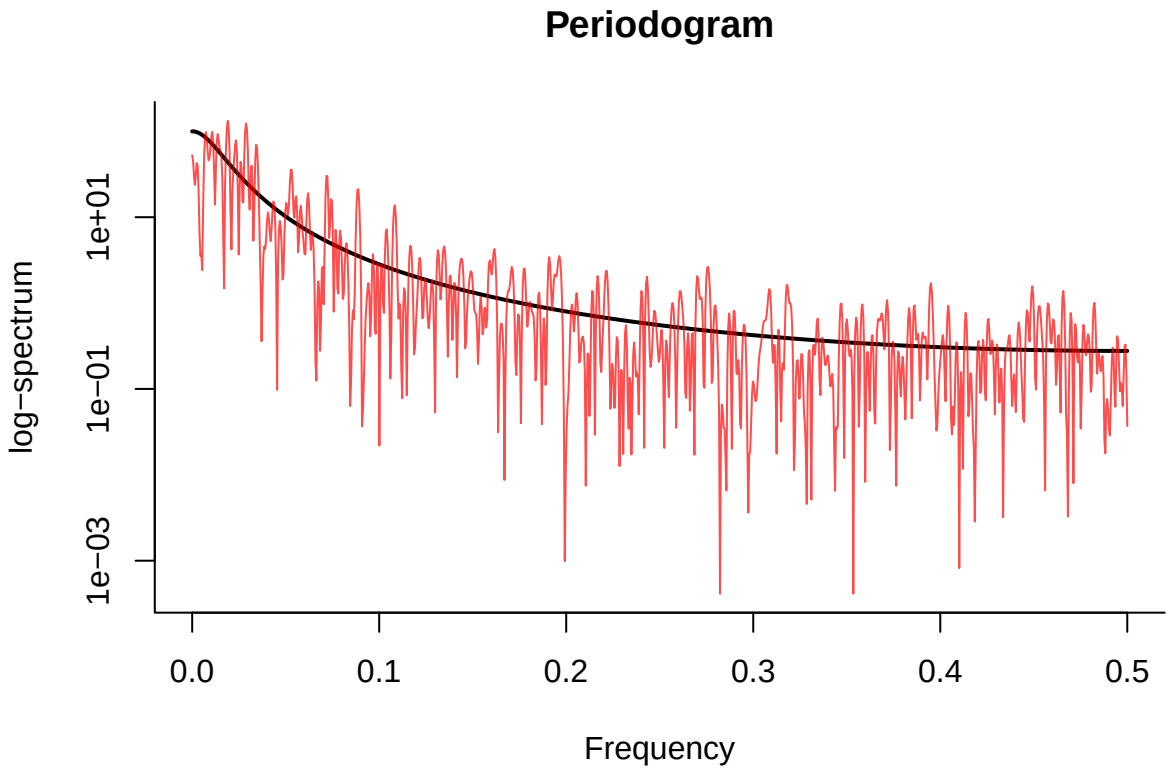
1. Simulate 500 observations from an AR(1) process with parameter values $\alpha \in \{0.1, 0.5, 0.9, 0.99\}$.
2. Repeat for MA processes of different orders. There is no restriction on the coefficients of the latter for stationarity, unlike the AR process.
3. Sample from an ARCH(1) process with Gaussian innovations and an ARCH(1) process with Student-t innovations with `df=4`. Look at the correlogram of the absolute residuals and the squared residuals.
4. The dataset `EuStockMarkets` contains the daily closing prices of major European stock indices. Type `?EuStockMarkets` for more details and `plot(EuStockMarkets)` to plot the four series (DAX, SMI, CAC and FTSE). Use `plot(ftse <- EuStockMarkets[,"FTSE"])` to plot the FTSE series and `plot(100*diff(log(ftse)))` to plot its daily log return. Play with the ARCH simulation functions to generate some similar processes.
5. Simulate a white noise series with trend t and $\cos(t)$, of the form $X_t = M_t + S_t + Z_t$, where $Z_t \sim \mathsf{N}(0, \sigma^2)$ for different values of σ^2. Analyze the log-periodogram and the (partial) correlograms. What happens if you forget to remove the trend?
6. Do the same for multiplicative model with lognormal margins, with structure $X_t = M_t S_t Z_t$.
7. For steps 5 and 6, plot the series and test the assumptions that they are white noise using the Ljung-Box test. *Note* you need to adjust the degrees of freedom when working with residuals from e.g. ARMA models.

1.5 Spectral analysis

We can compute the periodogram manually using the function `fft`. We pad the series with zero to increase the number of frequencies at which it is calculated (this does not impact the spectrum, because the new observations are zero). To fully take advantage of the fast Fourier transform (which will be formally defined later in the semester), we make sure the length of the padded series is divisible by low primes.

```
N <- 500  # number of data points
M <- 2048  # zeropadded length of series
freq <- seq(0, 0.5, by = 1/M)
alpha <- 0.9
x <- arima.sim(n = N, model = list(ar = alpha))

# theoretical spectrum
spec.thry <- TSA::ARMAspec(model = list(ar = alpha), freq = freq, plot = FALSE)

h.pgram <- rep(1/sqrt(N), N)  #periodogram taper / window
# prepared data
xh.pgram <- x * h.pgram
```

```
# calculate the periodogram manually with padding
spec.pgram <- abs(fft(c(xh.pgram, rep(0, M - N)))[1:(M/2 + 1)])^2
# Plot the series
plot(spec.thry$freq, spec.thry$spec, type = "l", log = "y", ylab = "log-spectrum",
    xlab = "Frequency", main = "Periodogram", lwd = 2, bty = "l", ylim = range(spec.pgram))
lines(freq, spec.pgram, col = rgb(1, 0, 0, 0.7))
```



The workhorse function for spectral analysis is `spectrum`, which computes and plots the periodogram on log scale with some default options. Note that `spectrum` by default subtract the mean from the series before estimating the spectral density and tapers the series (more later). To plot the cumulative periodogram, use `cpgram`. The latter shows the band for the Kolmogorov-Smirnov statistic. Note the presence of the 95 % confidence interval. The width of the center mark on it indicates the bandwidth.

1.6 Smoothing and detrending

We consider detrending of a temperature dataset from the monthly mean temperature from the Hadley center. We first download the dataset from the web. Typically, this file can be in a repository on your computer, or else you can provide an URL. Common formats include CSV (loaded using `read.csv`) and txt files (loaded via `read.table`. Be careful with the type, headers, missing values that are encoded using e.g. 999. Also note that **R** transforms strings into factors by default.

```
CET <- url("http://www.metoffice.gov.uk/hadobs/hadcet/cetml1659on.dat")
writeLines(readLines(CET, n = 10))
```

```
MONTHLY MEAN CENTRAL ENGLAND TEMPERATURE (DEGREES C)
1659-1973 MANLEY (Q.J.R.METEOROL.SOC., 1974)
1974 ON PARKER ET AL. (INT.J.CLIM., 1992)
```

```
PARKER AND HORTON (INT.J.CLIM., 2005)
```

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	YEAR
1659	3.0	4.0	6.0	7.0	11.0	13.0	16.0	16.0	13.0	10.0	5.0	2.0	8.87
1660	0.0	4.0	6.0	9.0	11.0	14.0	15.0	16.0	13.0	10.0	6.0	5.0	9.10
1661	5.0	5.0	6.0	8.0	11.0	14.0	15.0	15.0	13.0	11.0	8.0	6.0	9.78

```
cet <- read.table(CET, sep = "", skip = 6, header = TRUE, fill = TRUE, na.string = c(-99.99,
    -99.9))
names(cet) <- c(month.abb, "Annual")
## remove last row of incomplete data
cet <- cet[-nrow(cet), -ncol(cet)]
```

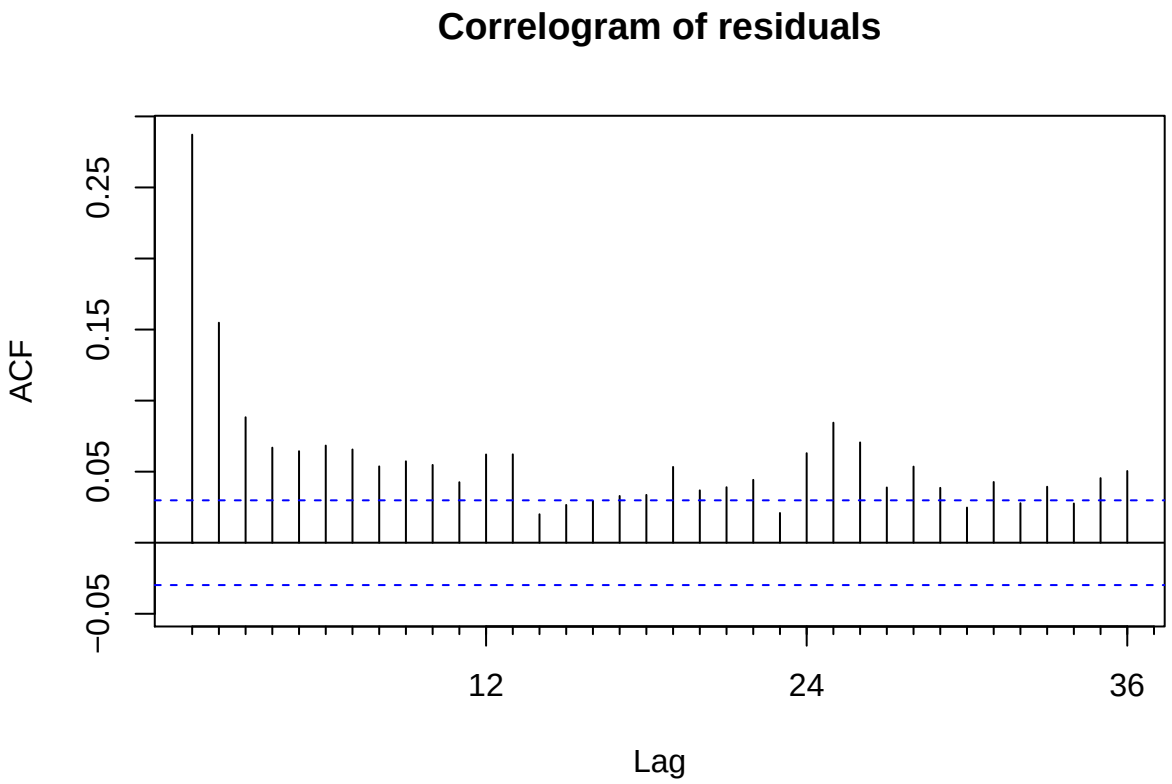
Now let us investigate the dataset in a regression context. Since it is an irregular time series, we use `zoo` rather than `ts`.

```
library(zoo)
library(lubridate)
library(forecast)
library(nlme)
```

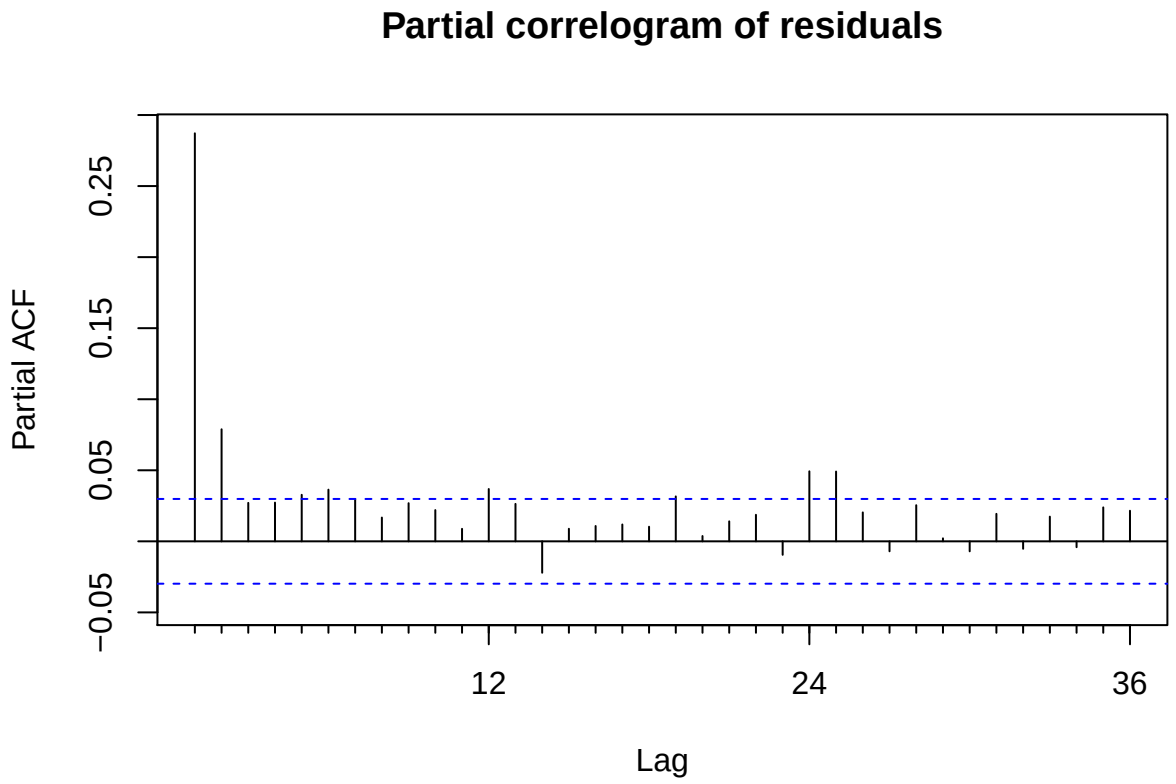
```
# Convert to time series object Create a time object using `seq`, then make
# into `yearmon` The latter has the same internal representation as `ts`
# with frequency
time <- zoo::as.yearmon(seq.Date(from = as.Date("1659/01/01"), length.out = prod(dim(cet)),
    by = "month"))
CET_ts <- zoo::zoo(c(t(cet)), time)
graphics.off()
plot(CET_ts, lwd = 0.2, ylab = expression(Temperature(degree * C)), main = "Monthly mean temperature\n
```

Now that we have extracted the data, we are now ready to try out with some models to explain seasonal variability as a function of covariates rather than via differencing. If your object is of class `ts`, the function `fourier` will do the Fourier basis of order `K` for you directly.

```
# Create Fourier basis manually
c1 <- cos(2 * pi * month(CET_ts)/12)
s1 <- sin(2 * pi * month(CET_ts)/12)
c2 <- cos(4 * pi * month(CET_ts)/12)
s2 <- sin(4 * pi * month(CET_ts)/12)

# Can also incorporate using fourier with `ts` objects.
ts1_a <- lm(CET_ts ~ time + fourier(CET_ts, K = 2))
ts1_b <- lm(CET_ts ~ seq.int(1, length(CET_ts)) + c1 + s1 + c2 + s2)
# Same fitted values, different regressors for the trend - see the design
# matrix head(ts1_a$model)
forecast::Acf(resid(ts1_a), main = "Correlogram of residuals")
```




```
forecast::Pacf(resid(ts1_b), main = "Partial correlogram of residuals")
```

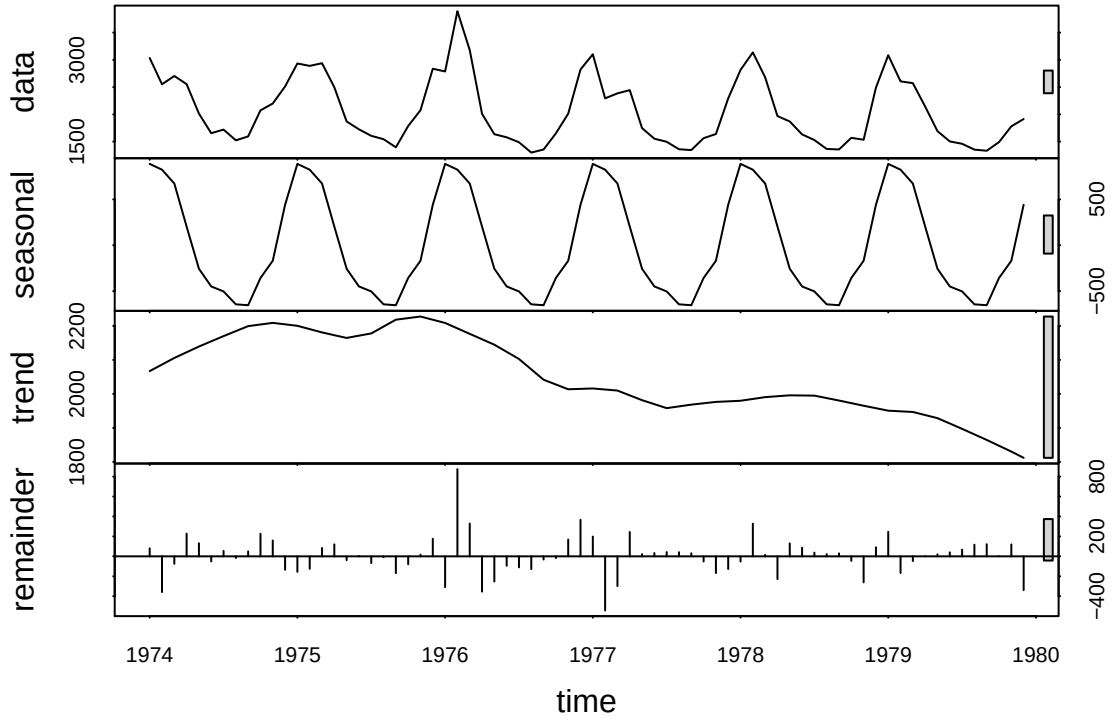


One could also replace the Fourier terms with seasonal dummies (possibly removing the intercept if 12 dummies are set). However, the use of Fourier terms, where appropriate, allows for more parsimonious modelling. Always

keep pairs of sine and cosine together.

The function `stl` decomposes a time series into seasonal trend and irregular components. We illustrate the use of the function on the `deaths` dataset.

```
seasonal_decomp_death <- stl(deaths, s.window = "periodic")
plot(seasonal_decomp_death)
```



1.6.1 Exercise 4: Mauna Loa Atmospheric CO_2 Concentration

1. Load and plot the CO_2 dataset from NOAA[14]. Pay special attention to the format, missing values, the handling of string and the description. Use `?read.table` for help, and look carefully at arguments `file`, `sep`, `na.strings`, `skip` and `stringsAsFactors`. From now on, we will work with the complete series (termed interpolated in the description).
2. Try removing the trend using a linear model. Plot the residuals against month of the year.
3. Remove the trend and the periodicity with a Fourier basis (with period 12). Be sure to include both `sin` and `cos` terms together. Recall that the standard Wald tests for the coefficients is not valid in the presence of autocorrelation! You could also use `poly` or `splines::bs` to fit polynomials or splines to your series.
4. Plot the lagged residuals. Are there evidence of correlation?
5. Use the function `filter` to smooth the series using a 12 period moving average.
6. Inspect the spectrum of the raw series and of the smoothed version.
7. Inspect the spectrum of the detrended raw series.
8. Test for stationarity of the deseasonalized and detrended residuals using the KPSS test viz. `tseries::kpss.test`.
9. Use the `decompose` and the `stl` functions to obtain residuals.
10. Plot the (partial) correlogram for both decomposition and compare them with the output of the linear model.

[14] ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_mm_mlo.txt

1.7 Solutions to Exercises

1.7.1 Solutions 1: Beaver temperature

1. Load the `beav2` data from the library `MASS`.
2. Examine the data frame using `summary`, `head`, `tail`. Query the help with `?beav2` for a description of the dataset
3. Transform the temperature data into a time series object and plot the latter.
4. Fit a linear model using `lm` and the variable `activ` as factor, viz. `lin_mod <- lm(temp~as.factor(activ), data=beav2)`. Overlay the means on your plot with `lines(fitted(lin_mod))` replacing `lin_mod` with your `lm` result.
5. Inspect the residuals (`resid(lin_mod)`) and determine whether there is any evidence of trend or seasonality.
6. Look at a quantile-quantile (Q-Q) plot to assess normality. You can use the command `qqnorm` if you don't want to transform manually the residuals with `qqline` or use `plot(lin_mod, which=2)`.
7. Plot the lag-one residuals at time t and $t-1$. Is the dependence approximately linear?

```
data(beav2, package = "MASS")
`?`(MASS::beav2)
beav2$hours <- with(beav2, 24 * (day - 307) + trunc(time/100) + (time%%100)/60)
summary(beav2)
```

```
      day             time           temp           activ       
 Min.   :307.0   Min.   :   0   Min.   :36.58   Min.   :0.00  
 1st Qu.:307.0   1st Qu.:1128   1st Qu.:37.15   1st Qu.:0.00  
 Median :307.0   Median :1535   Median :37.73   Median :1.00  
 Mean   :307.1   Mean   :1446   Mean   :37.60   Mean   :0.62  
 3rd Qu.:307.0   3rd Qu.:1942   3rd Qu.:37.98   3rd Qu.:1.00  
 Max.   :308.0   Max.   :2350   Max.   :38.35   Max.   :1.00  
     hours      
 Min.   : 9.50  
 1st Qu.:13.62  
 Median :17.75  
 Mean   :17.75  
 3rd Qu.:21.88  
 Max.   :26.00  
```

```
head(beav2)
```

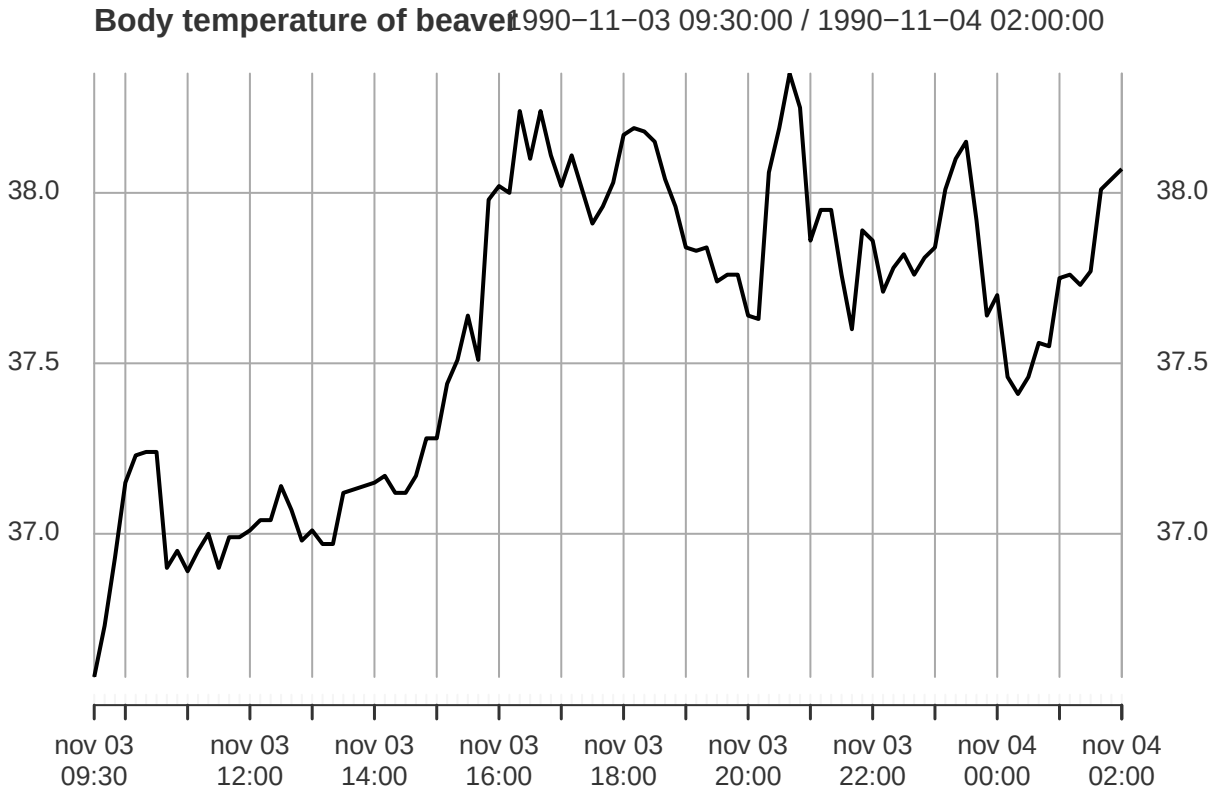
```
  day time  temp activ     hours
1 307  930 36.58     0  9.500000
2 307  940 36.73     0  9.666667
3 307  950 36.93     0  9.833333
4 307 1000 37.15     0 10.000000
5 307 1010 37.23     0 10.166667
6 307 1020 37.24     0 10.333333
```

```
tail(beav2)
```

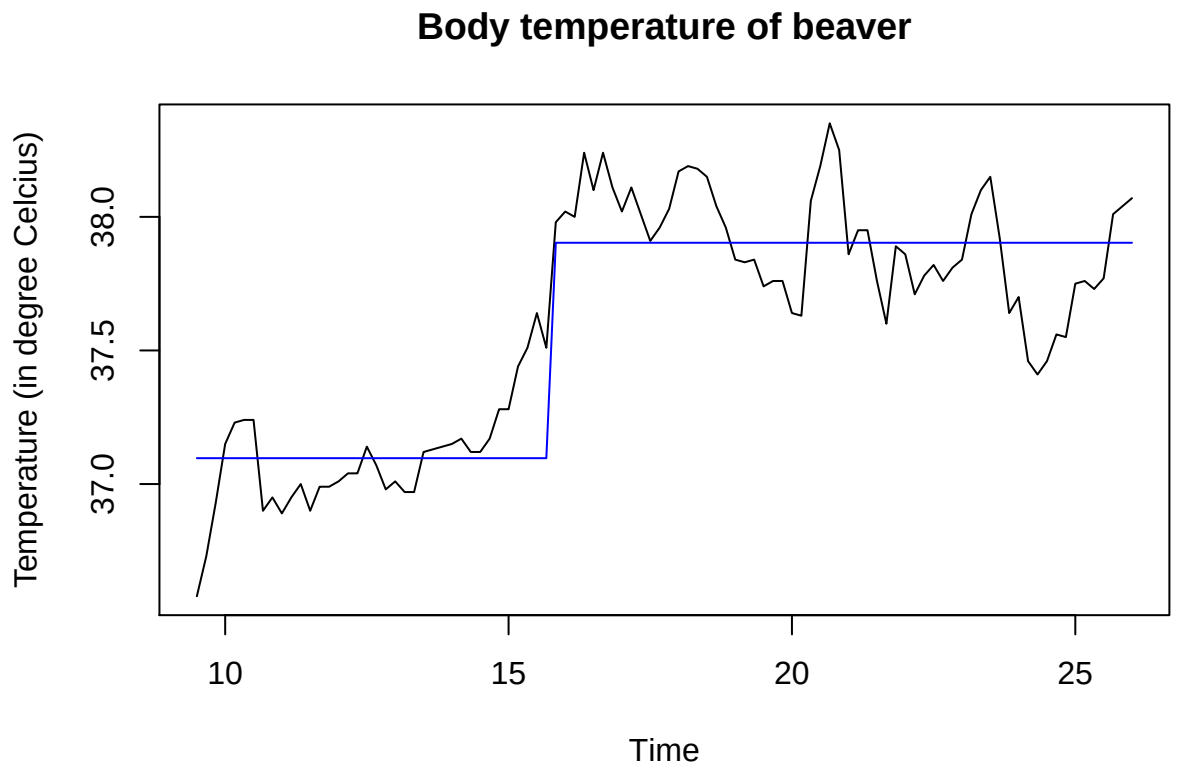
```
    day time  temp activ    hours
95  308  110 37.76     1 25.16667
96  308  120 37.73     1 25.33333
```

```
97  308  130 37.77     1 25.50000
98  308  140 38.01     1 25.66667
99  308  150 38.04     1 25.83333
100 308  200 38.07     1 26.00000
```

```
# Fancy time series object
hours <- seq(ISOdatetime(1990, 11, 3, 9, 30, 0), ISOdatetime(1990, 11, 4, 2,
    0, 0), by = (60 * 10))
plot(xts::xts(beav2[, "temp"], hours), main = "Body temperature of beaver",
    ylab = "Temperature (in degree Celcius)")
```

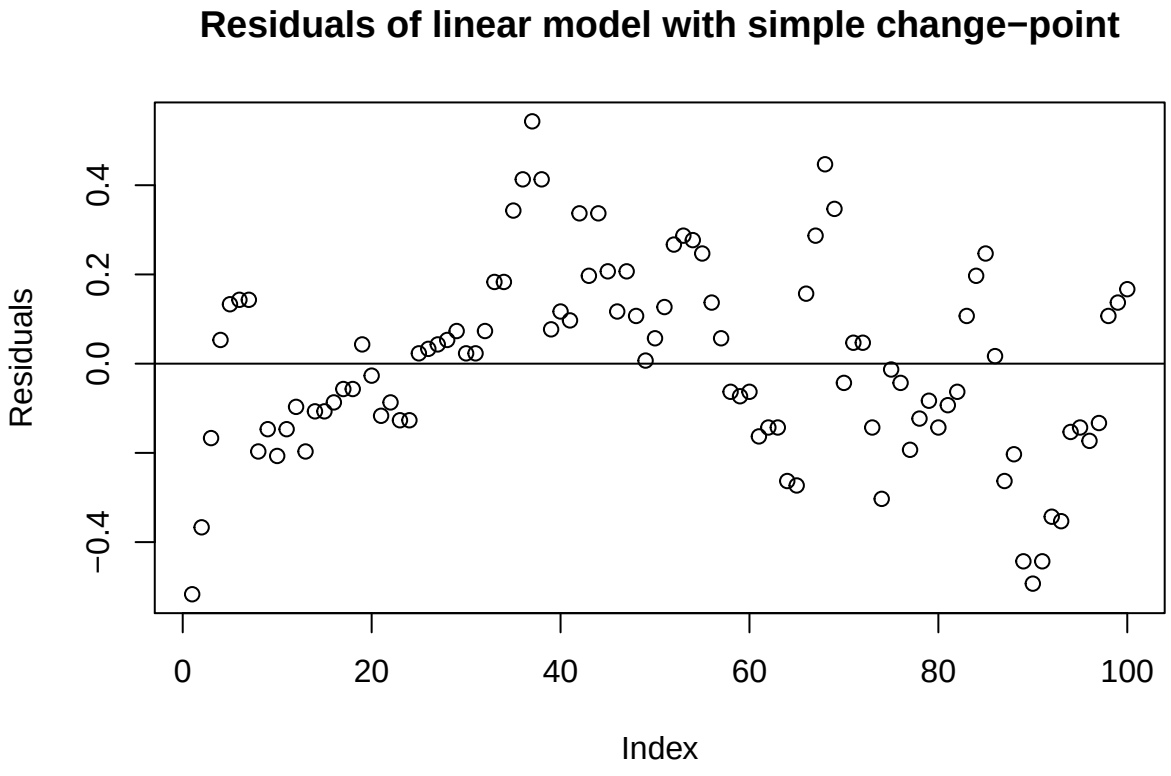



```
# Vanilla ts - works ok for regular time series
temp <- ts(beav2[, "temp"], start = 9.5, frequency = 6)
plot(temp, main = "Body temperature of beaver", ylab = "Temperature (in degree Celcius)")
lin_mod <- lm(temp ~ as.factor(activ), data = beav2)
lines(beav2[, "hours"], fitted(lin_mod), col = "blue")
```

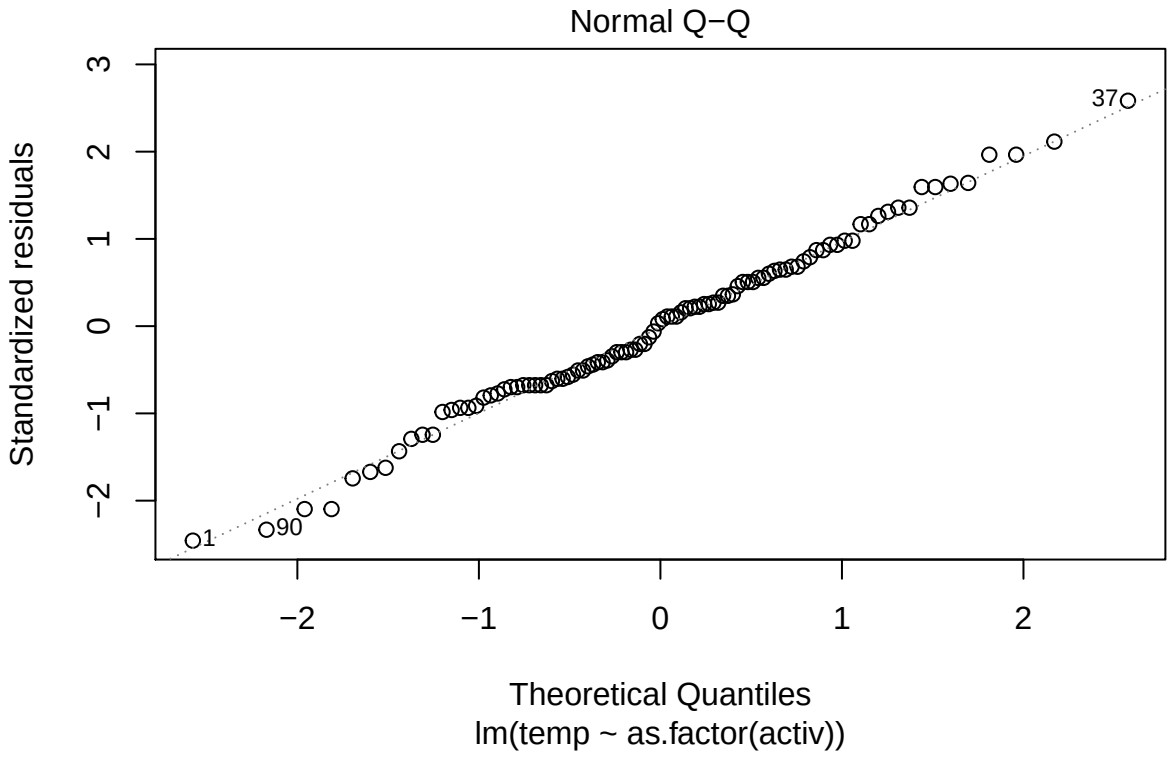



```
# Some trend remaining in the first part, before time
```

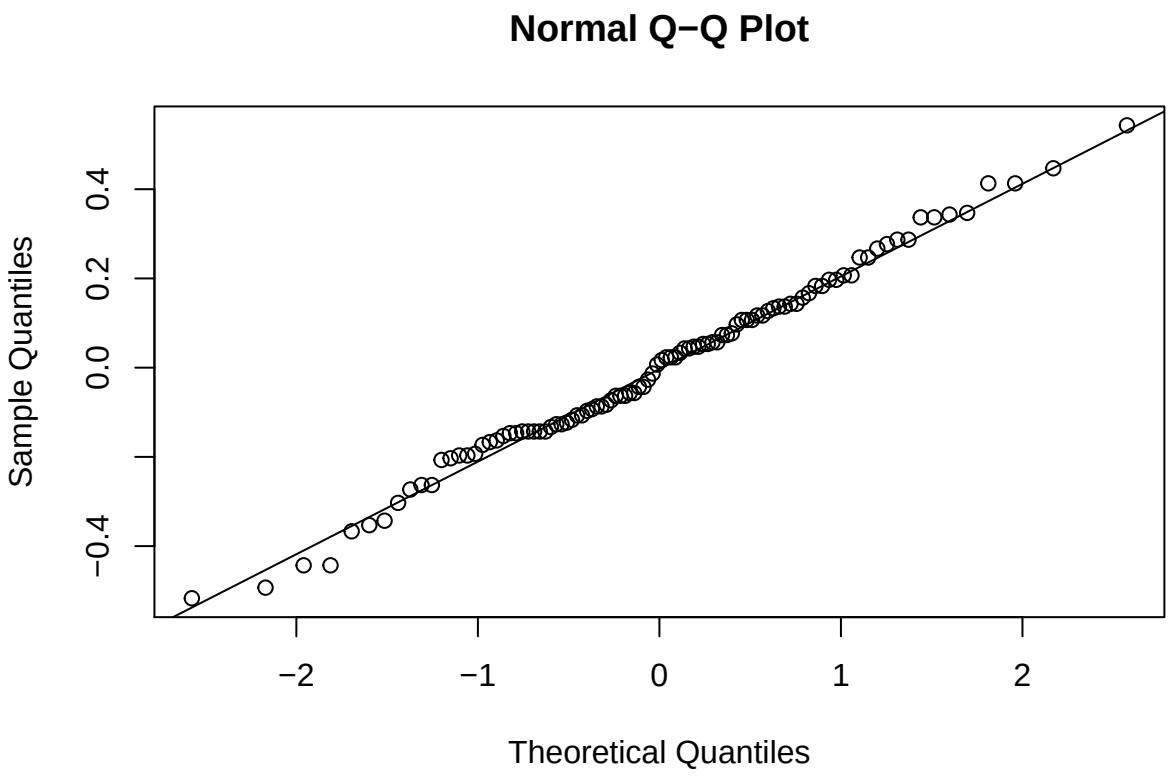
```
plot(residuals(lin_mod), ylab = "Residuals", main = "Residuals of linear model with simple change-point
abline(h = 0)
```



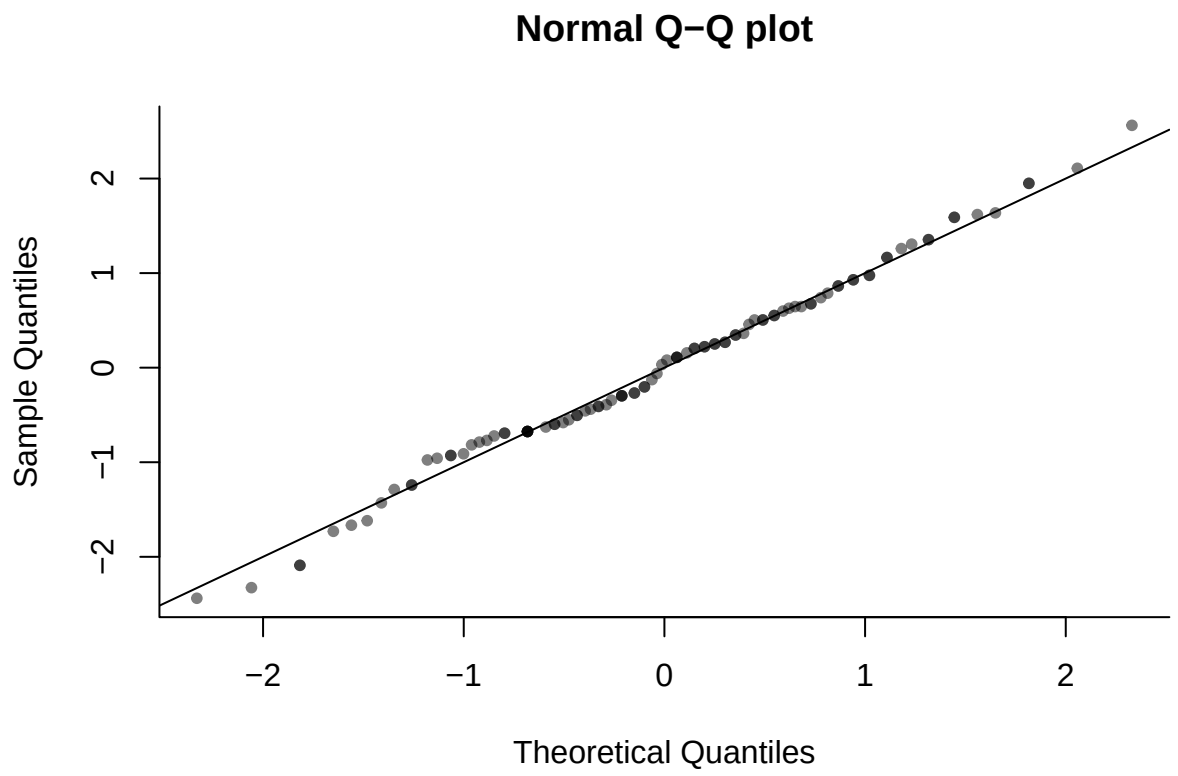
```r
# Q-Q plot (1) with output from lm
plot(lin_mod, which = 2)
```



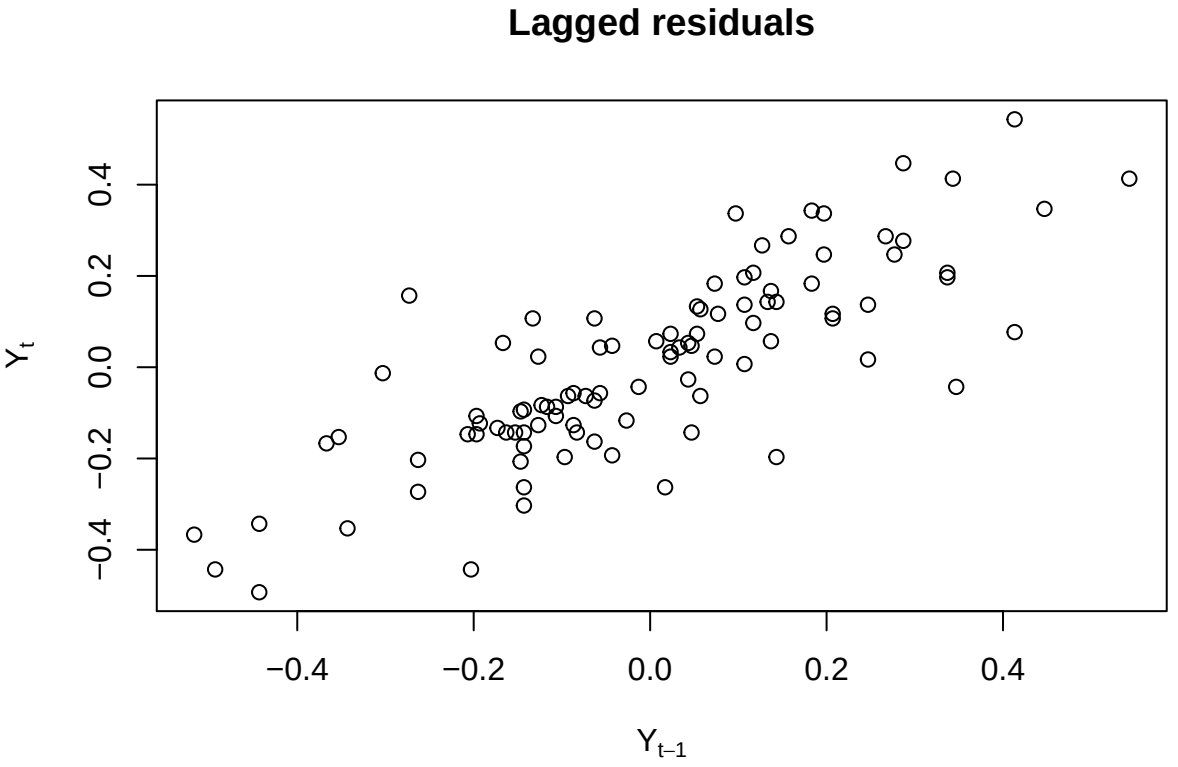

```r
# (2) with qqnorm and tentative line with quartiles
qqnorm(residuals(lin_mod))
qqline(residuals(lin_mod))
```




```r
# (3) standardized manually
res <- residuals(lin_mod)
plot(qnorm(rank(res)/(length(res) + 1)), res/sd(res), pty = "s", bty = "l",
    xlab = "Theoretical Quantiles", ylab = "Sample Quantiles", main = "Normal Q-Q plot",
    pch = 20, col = rgb(0, 0, 0, 0.5))
abline(a = 0, b = 1)
```




```
plot(res[-length(res)], res[-1], xlab = expression(Y[t - 1]), ylab = expression(Y[t]),
    main = "Lagged residuals")
```



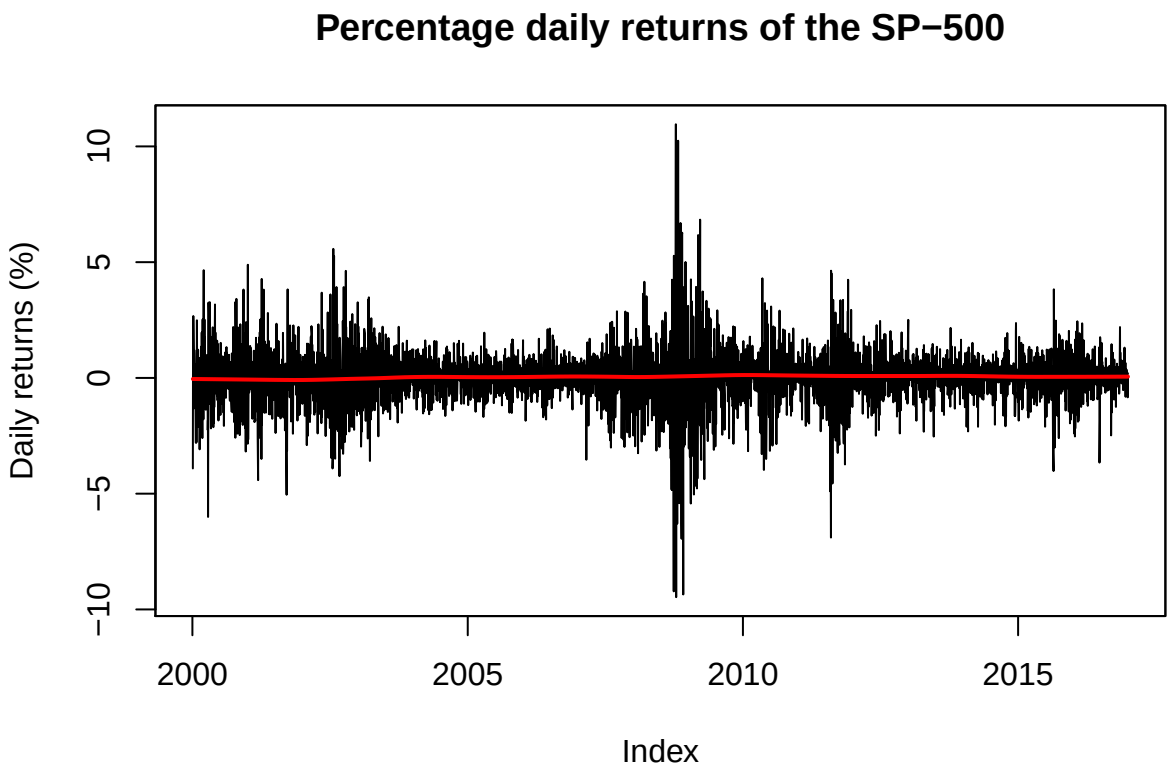
1.7.2 Solutions 2: SP500 daily returns

1. Download the dataset using the following command
2. Obtain the daily percent return series and plot the latter against time.
3. With the help of graphs, discuss evidences of seasonality and nonstationarity. Are there seasons of returns?
4. Plot the (partial) correlogram of both the raw and the return series. Try the acf with `na.action=na.pass` and without (by e.g. converting the series to a vector using `as.vector`. Comment on the impact of ignoring time stamps.
5. Plot the (partial) correlogram of the absolute value of the return series and of the squared return series. What do you see?

```
sp500 <- tseries::get.hist.quote(instrument = "^GSPC", start = "2000-01-01",
    end = "2016-12-31", quote = "AdjClose", provider = "yahoo", origin = "1970-01-01",
    compression = "d", retclass = "zoo")
```

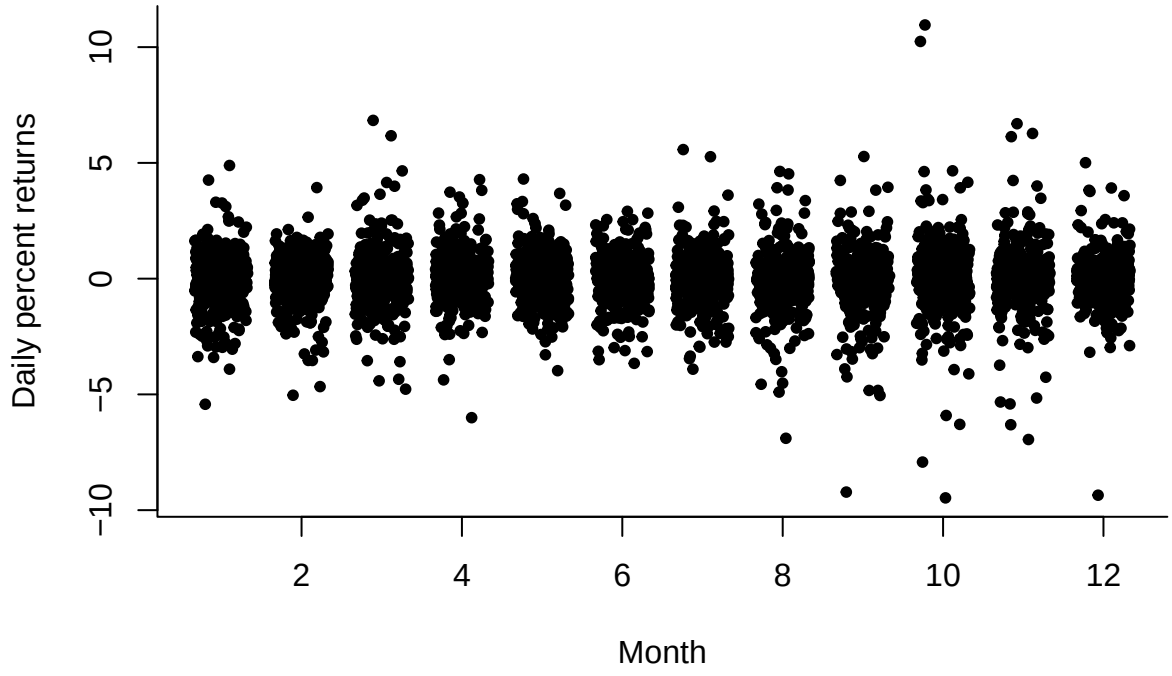
```
time series starts 2000-01-03
time series ends   2016-12-30
```

```
library(xts)
library(lubridate)
# Daily return in percentage
spret <- 100 * diff(log(sp500))
plot(spret, ylab = "Daily returns (%)", main = "Percentage daily returns of the SP-500")
# local trend to see if there is any evidence of non-zero trend
lines(index(spret), lowess(spret, f = 1/5)$y, col = 2, lwd = 2)
```

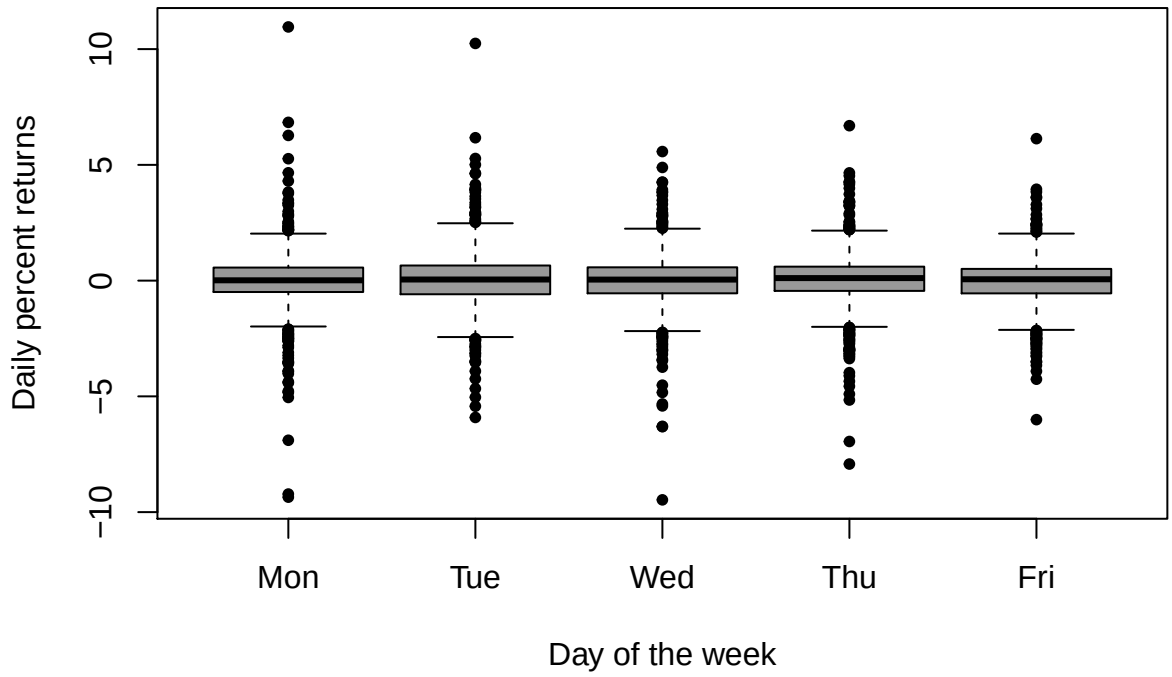



```
# Volatility as function of month
plot(jitter(month(spret), amount = 1/3), spret, pch = 20, ylab = "Daily percent returns",
```

```
    xlab = "Month", bty = "l")
```

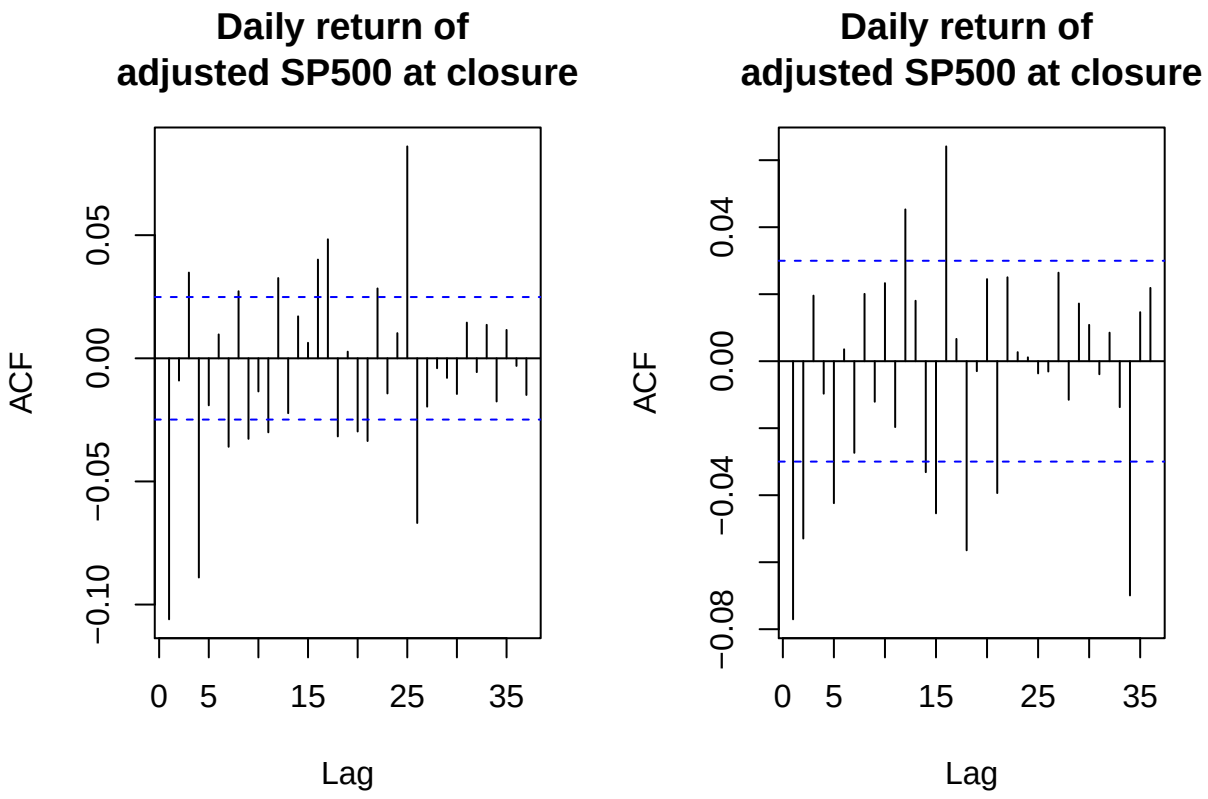



```
boxplot(as.vector(spret) ~ factor(wday(spret), labels = c("Mon", "Tue", "Wed",
    "Thu", "Fri")), xlab = "Day of the week", pch = 20, col = rgb(0, 0, 0, 0.4),
    ylab = "Daily percent returns", bty = "l")
```




```
# More uncertainty in March-May and August-November Some more extremes early
# in the week
```

```
par(mfrow = c(1, 2))
title_sp <- "Daily return of \nadjusted SP500 at closure"
TSA::acf(spret, na.action = na.pass, main = title_sp)
TSA::acf(na.omit(as.vector(spret)), main = title_sp)
```




```
dev.off()
```

```
null device
          1
```

```
pacf(spret, na.action = na.pass, main = title_sp)

# (P)ACF of absolute value of daily returns
TSA::acf(abs(spret), na.action = na.pass, main = title_sp)
pacf(abs(spret), na.action = na.pass, main = title_sp)
# (P)ACF of squared daily returns
TSA::acf(I(spret^2), na.action = na.pass, main = title_sp)
pacf(I(spret^2), na.action = na.pass, main = title_sp)
```

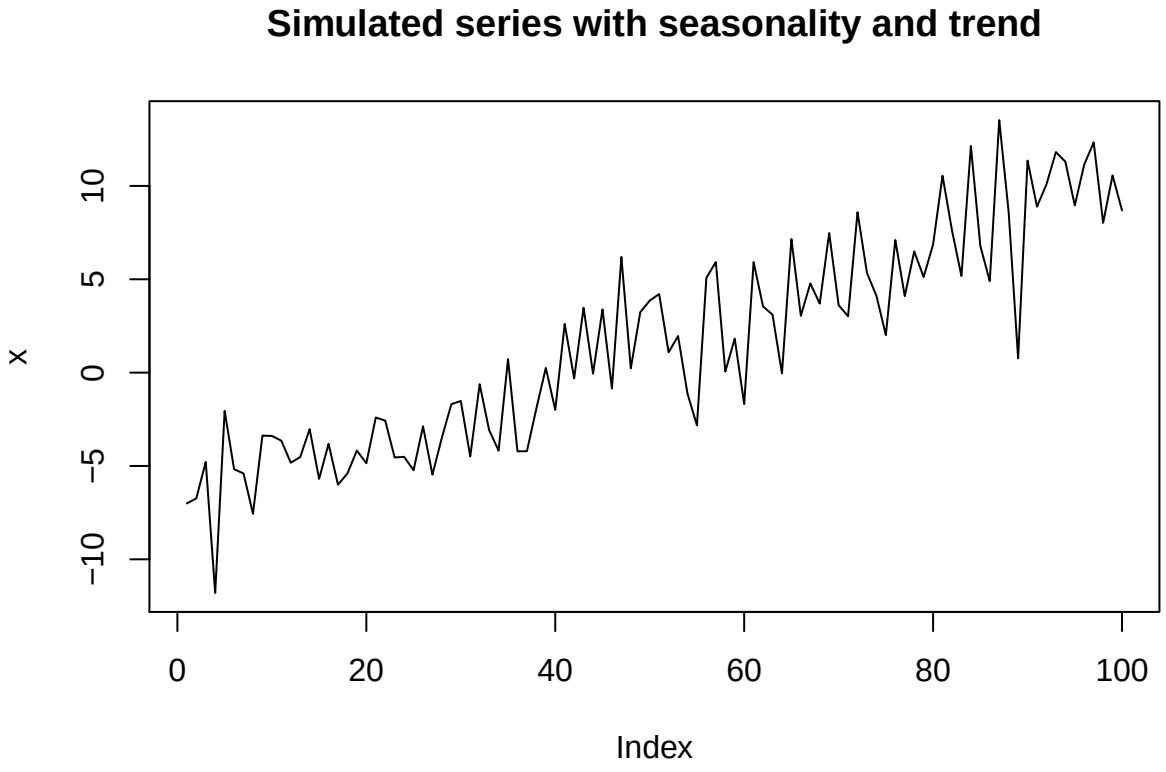
1.7.3 Solutions 3: Simulated data

The first 5 parts of the question are straightforward and left to the reader.

1. Simulate 500 observations from an AR(1) process with parameter values $\alpha \in \{0.1, 0.5, 0.9, 0.99\}$.
2. Repeat for MA processes of different orders. There is no restriction on the coefficients of the latter for stationarity, unlike the AR process.

3. Sample from an ARCH(1) process with Gaussian innovations and an ARCH(1) process with Student-t innovations with `df=4`. Look at the correlogram of the absolute residuals and the squared residuals.
4. The dataset `EuStockMarkets` contains the daily closing prices of major European stock indices. Type `?EuStockMarkets` for more details and `plot(EuStockMarkets)` to plot the four series (DAX, SMI, CAC and FTSE). Use `plot(ftse <- EuStockMarkets[,"FTSE"])` to plot the FTSE series and `plot(100*diff(log(ftse)))` to plot its daily log return. Play with the ARCH simulation functions to generate some similar processes.
5. Simulate a white noise series with trend t and $\cos(t)$, of the form $X_t = M_t + S_t + Z_t$, where $Z_t \sim \mathsf{N}(0, \sigma^2)$ for different values of σ^2. Analyze the log-periodogram and the (partial) correlograms. What happens if you forget to remove the trend?
6. Do the same for multiplicative model with lognormal margins, with structure $X_t = M_t S_t Z_t$.
7. For steps 5 and 6, plot the series and test the assumptions that they are white noise using the Ljung-Box test. *Note* you need to adjust the degrees of freedom when working with residuals from e.g. ARMA models.

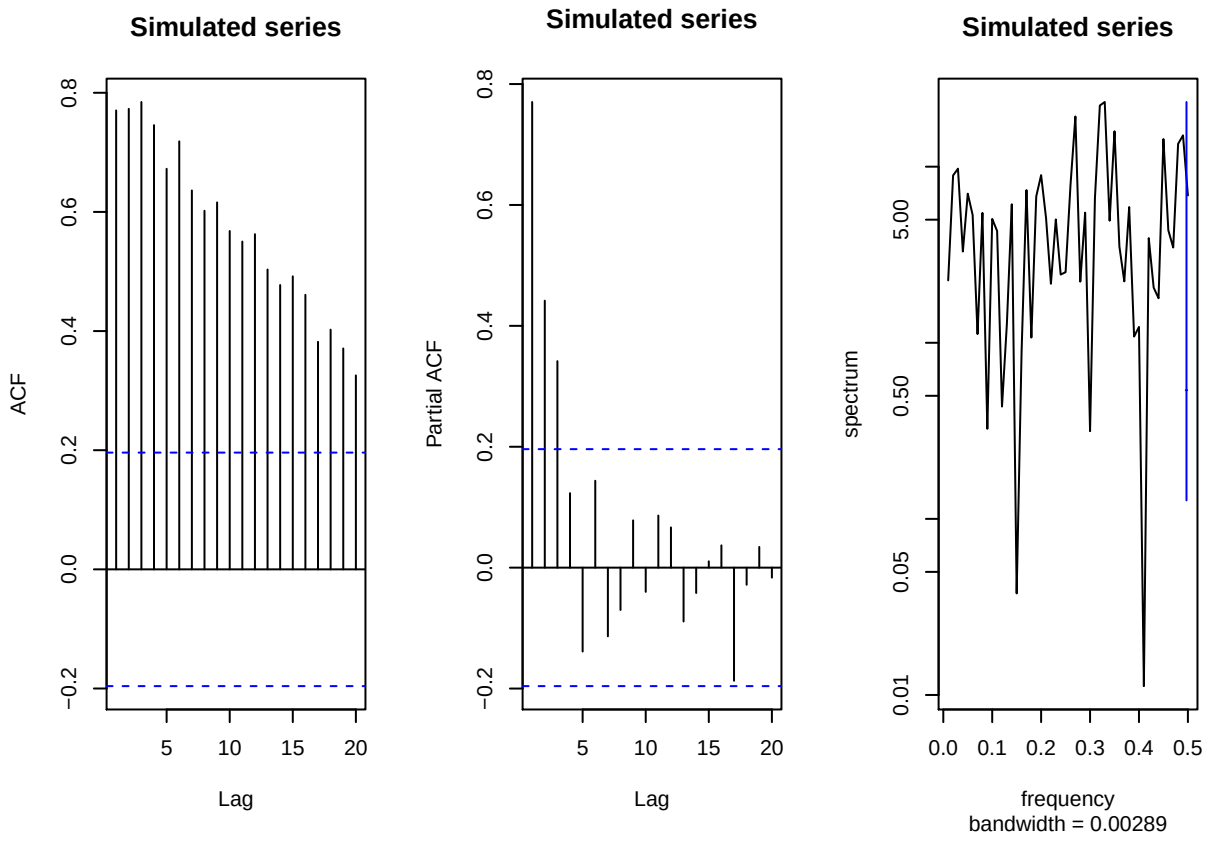
```
n <- 100
tim <- scale(1:n)
x <- 5 * tim + cos(2 * pi * tim/n) + rnorm(n, sd = 3)
plot(x, type = "l", main = "Simulated series with seasonality and trend")
```



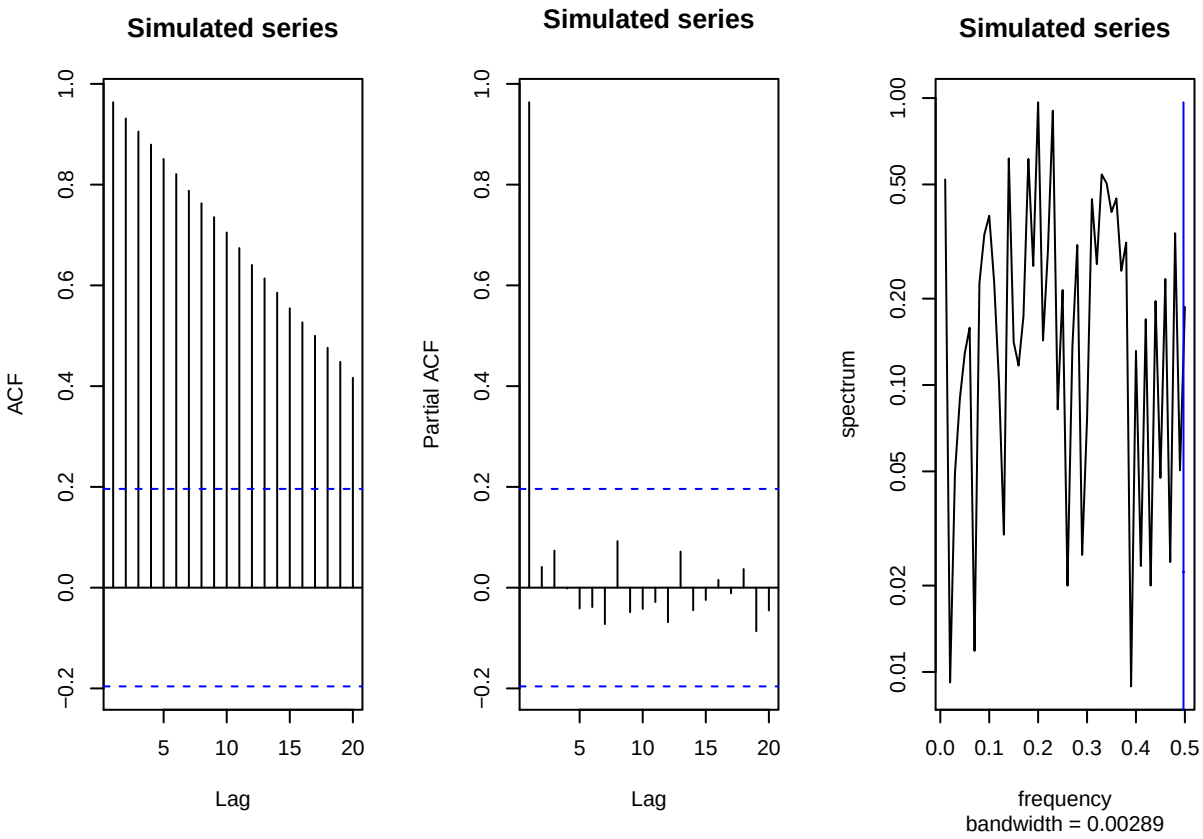

```
Box.test(x, type = "L")
```

```
    Box-Ljung test

data:  x
X-squared = 61.14, df = 1, p-value = 5.329e-15
```

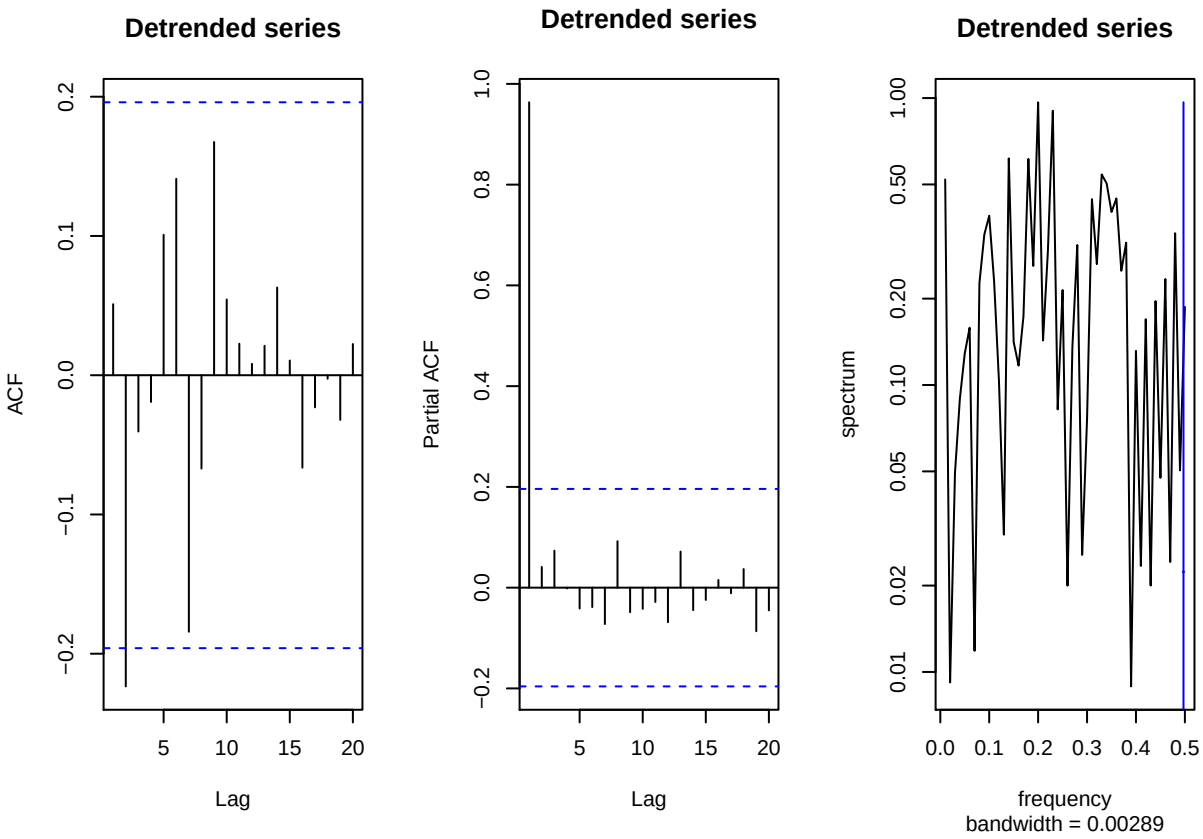
```r
par(mfrow = c(1, 3))  # plots side by side
TSA::acf(x, main = "Simulated series")
pacf(x, main = "Simulated series")
spectrum(x, main = "Simulated series")
```



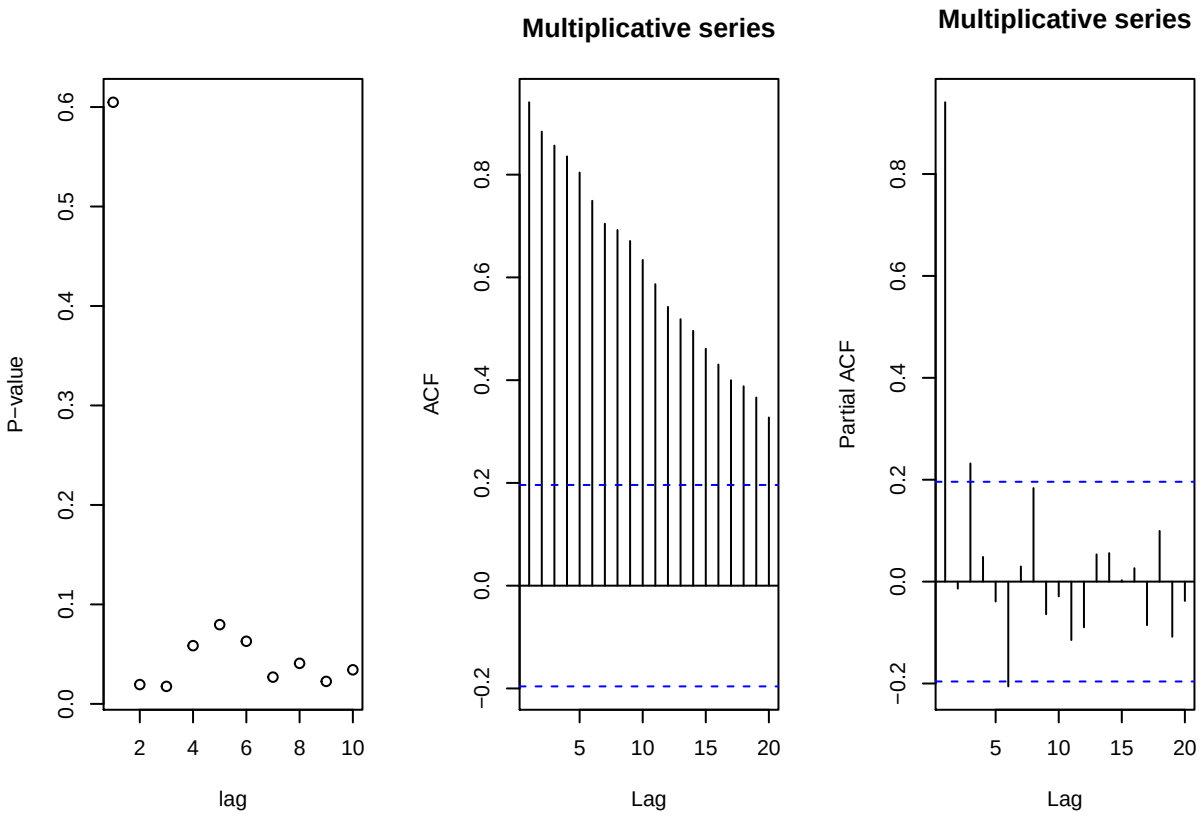

```r
# Nothing in spectrum, persistence in the (p)acf
x <- 5 * tim + cos(2 * pi * tim/n) + rnorm(n, sd = 0.5)
TSA::acf(x, main = "Simulated series")
pacf(x, main = "Simulated series")
spectrum(x, main = "Simulated series")
```




```
# Worst if the signal is strong relative to noise

y <- residuals(lm(x ~ 1 + tim))
TSA::acf(y, main = "Detrended series")
pacf(x, main = "Detrended series")
spectrum(y, main = "Detrended series")
```




```r
# The cosine still induces some lag-one dependence in pacf
plot(1:10, sapply(1:10, function(i) {
    Box.test(y, lag = i, type = "Ljung", fitdf = min(i - 1, 2))$p.value
}), ylab = "P-value", xlab = "lag")
# These low p-values at large lags are due to the cosine term

mult <- exp(scale(x))
TSA::acf(mult, main = "Multiplicative series")
pacf(mult, main = "Multiplicative series")
```




```
spectrum(mult, main = "Multiplicative series")
Box.test(mult, type = "L")
```

```
    Box-Ljung test

data:  mult
X-squared = 91.204, df = 1, p-value < 2.2e-16
```

```
graphics.off()
# Now large impact on spectrum, and nonlinear features!
plot(mult, main = "Multiplicative series with lognormal margins", ylab = "",
    xlab = "Time")
# Note that decompose has an option type='multiplicative' for seasonal
# components
```

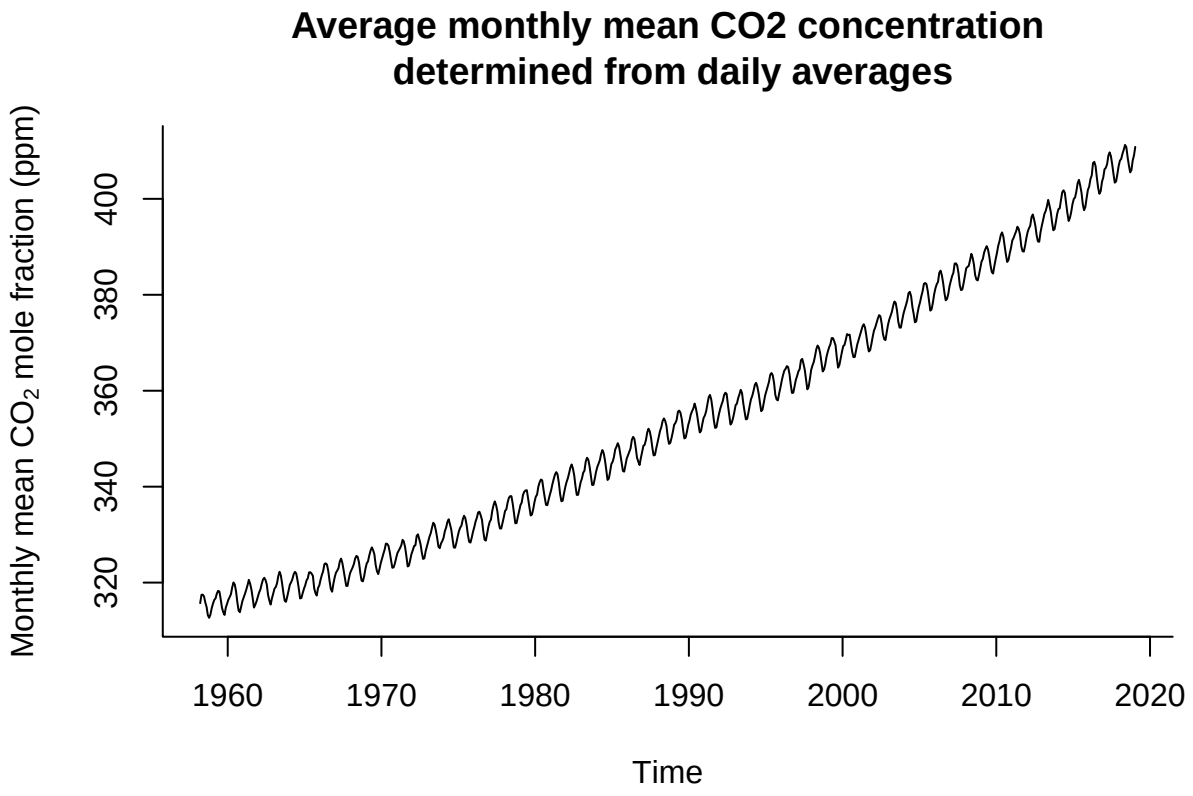
1.7.4 Solutions 4: Mauna Loa Atmospheric CO_2 Concentration

1. Load and plot the CO_2 dataset from NOAA[15]. Pay special attention to the format, missing values, the handling of string and the description. Use `?read.table` for help, and look carefully at arguments `file`, `sep`, `na.strings`, `skip` and `stringsAsFactors`. From now on, we will work with the complete series (termed interpolated in the description).
2. Try removing the trend using a linear model. Plot the residuals against month of the year.

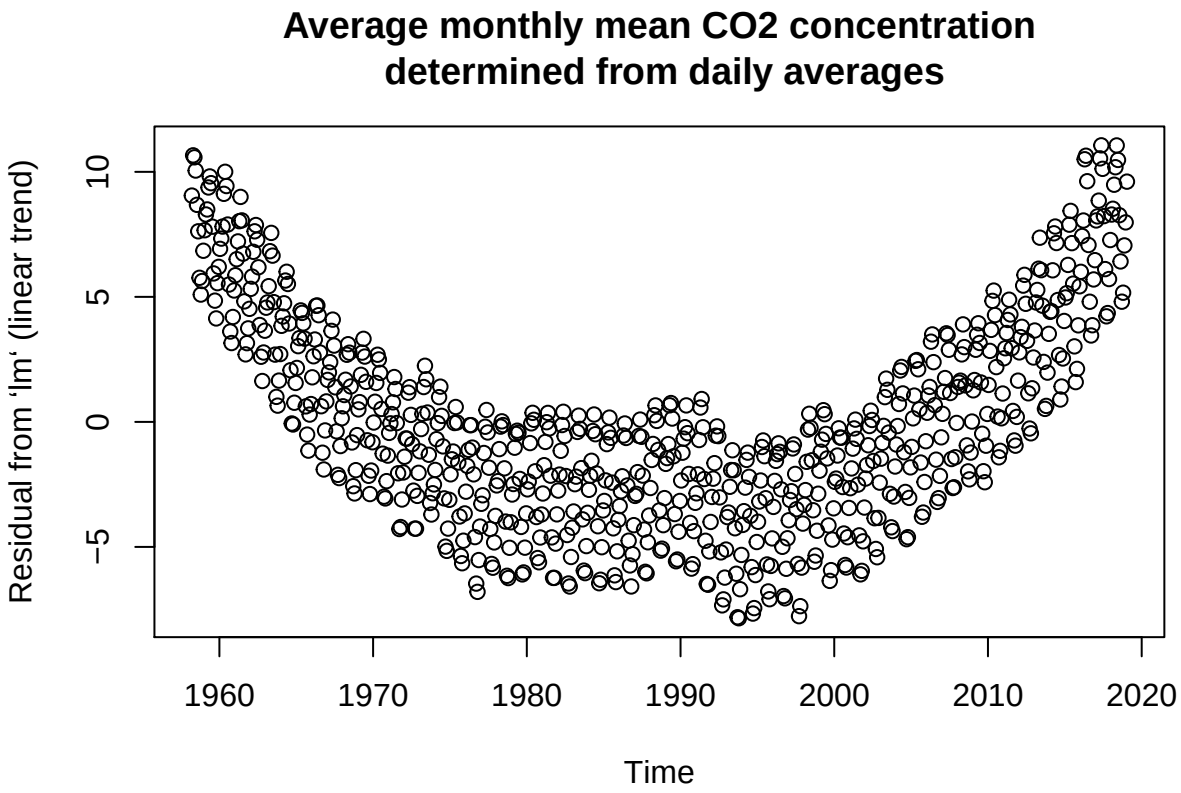
[15] ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_mm_mlo.txt

3. Remove the trend and the periodicity with a Fourier basis (with period 12). Be sure to include both `sin` and `cos` terms together. Recall that the standard Wald tests for the coefficients is not valid in the presence of autocorrelation! You could also use `poly` or `splines::bs` to fit polynomials or splines to your series.
4. Plot the lagged residuals. Are there evidence of correlation?
5. Use the function `filter` to smooth the series using a 12 period moving average.
6. Inspect the spectrum of the raw series and of the smoothed version.
7. Inspect the spectrum of the detrended raw series.
8. Test for stationarity of the deseasonalized and detrended residuals using the KPSS test viz. `tseries::kpss.test`.
9. Use the `decompose` and the `stl` functions to obtain residuals.
10. Plot the (partial) correlogram for both decomposition and compare them with the output of the linear model.

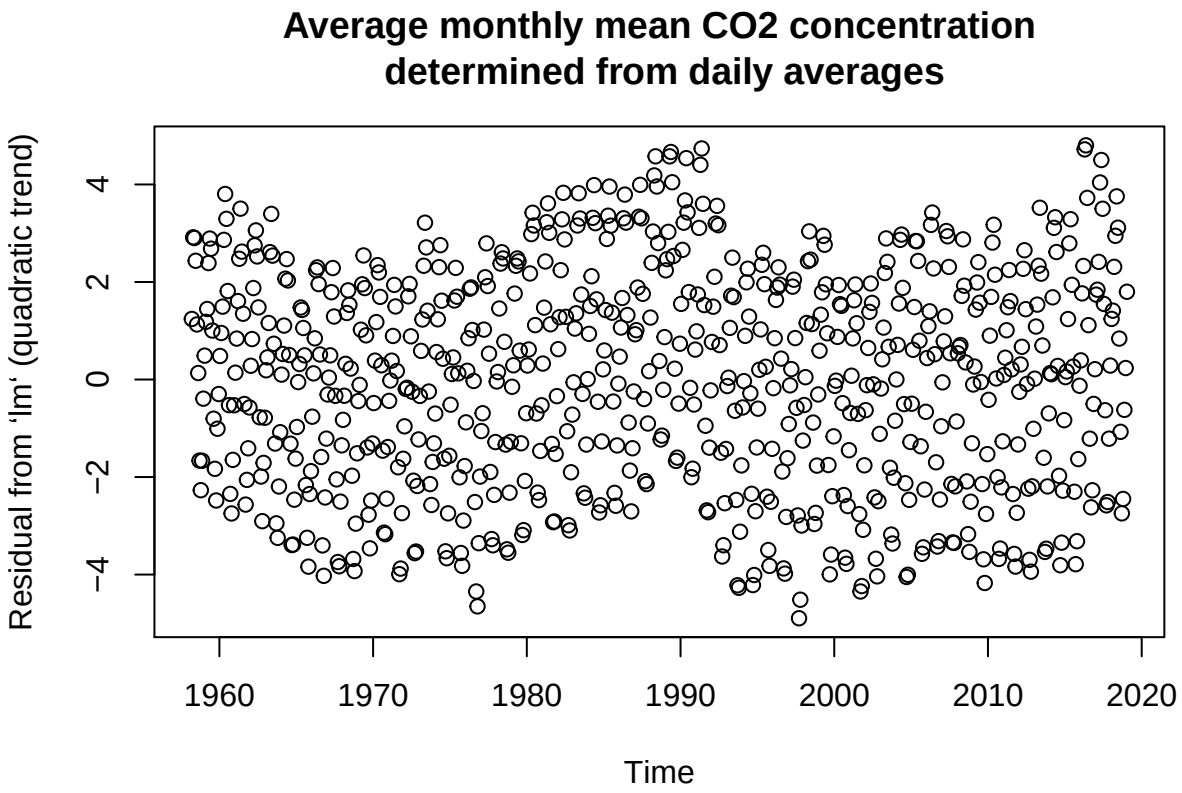
```
# Because of comment.char=#, all the first lines are skipped But we lose the
# header
co2 <- read.table(file = "ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_mm_mlo.txt",
    na.strings = "-99.99", stringsAsFactors = FALSE, col.names = c("year", "month",
        "time", "average", "interpolated", "trend", "monthly_mean"))
# install.packages('stlplus') #this package offers a version of stl that
# deals with NAs

ycap <- expression(paste("Monthly mean ", CO[2], " mole fraction (ppm)"))
mcap <- "Average monthly mean CO2 concentration\n determined from daily averages"
lin_mod <- lm(data = co2, interpolated ~ time)
with(co2, plot(interpolated ~ time, type = "l", xlab = "Time", bty = "l", ylab = ycap,
    main = mcap))
```



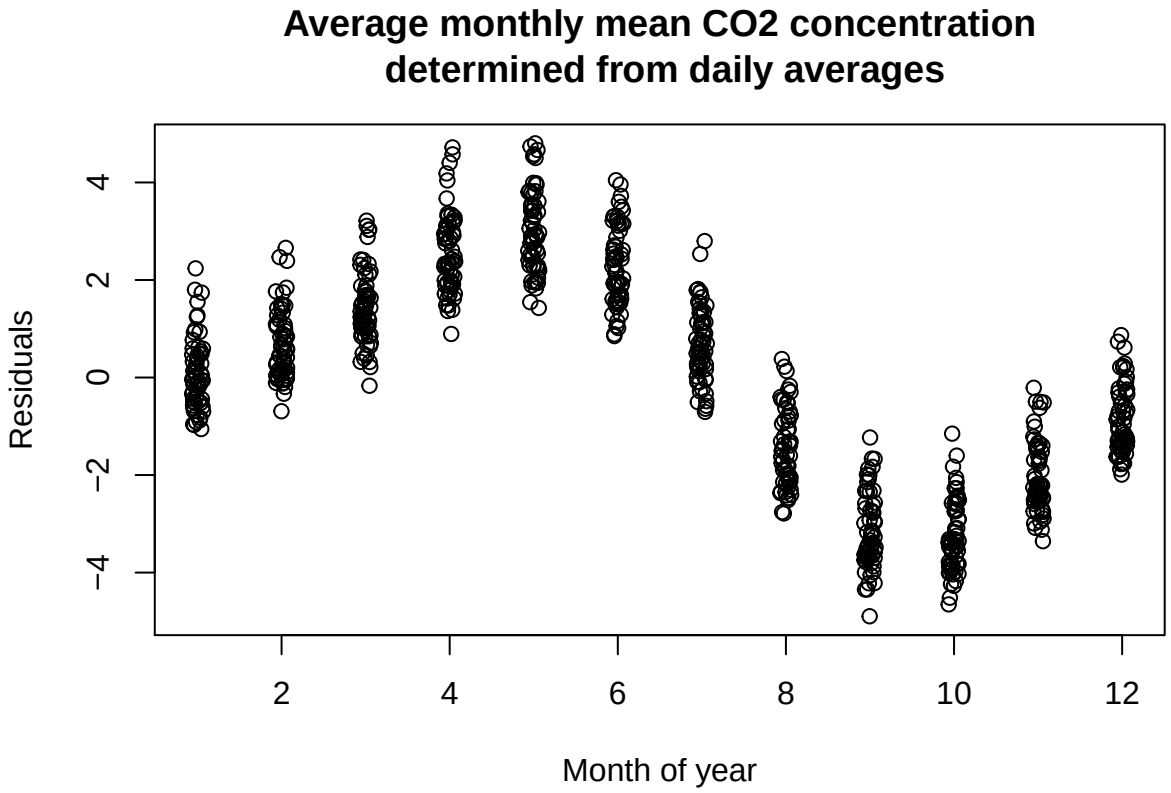

```
plot(co2$time, residuals(lin_mod), ylab = "Residual from `lm` (linear trend)",
    xlab = "Time", main = mcap)
```



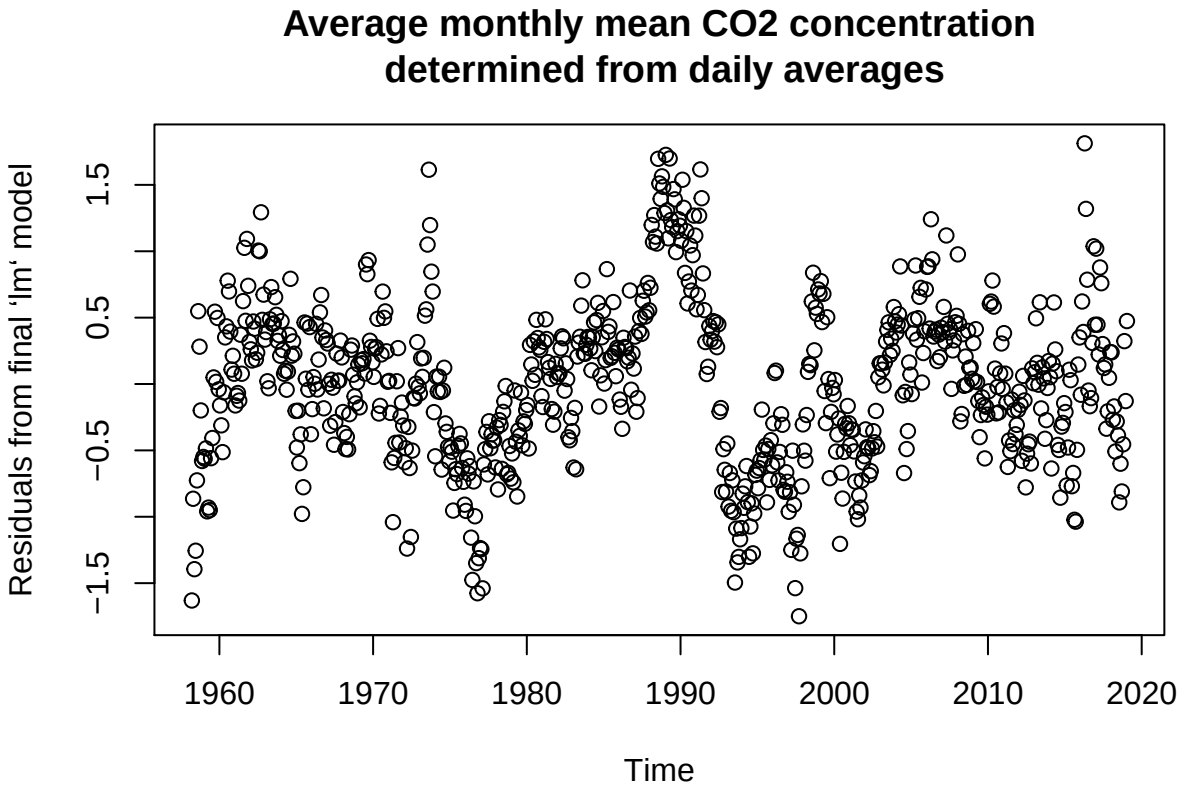

```
lin_mod <- update(lin_mod, . ~ . + I(time^2))
# same as lm(data=co2, interpolated ~ poly(time, degree = 2))
plot(co2$time, residuals(lin_mod), main = mcap, ylab = "Residual from `lm` (quadratic trend)",
    xlab = "Time")
```



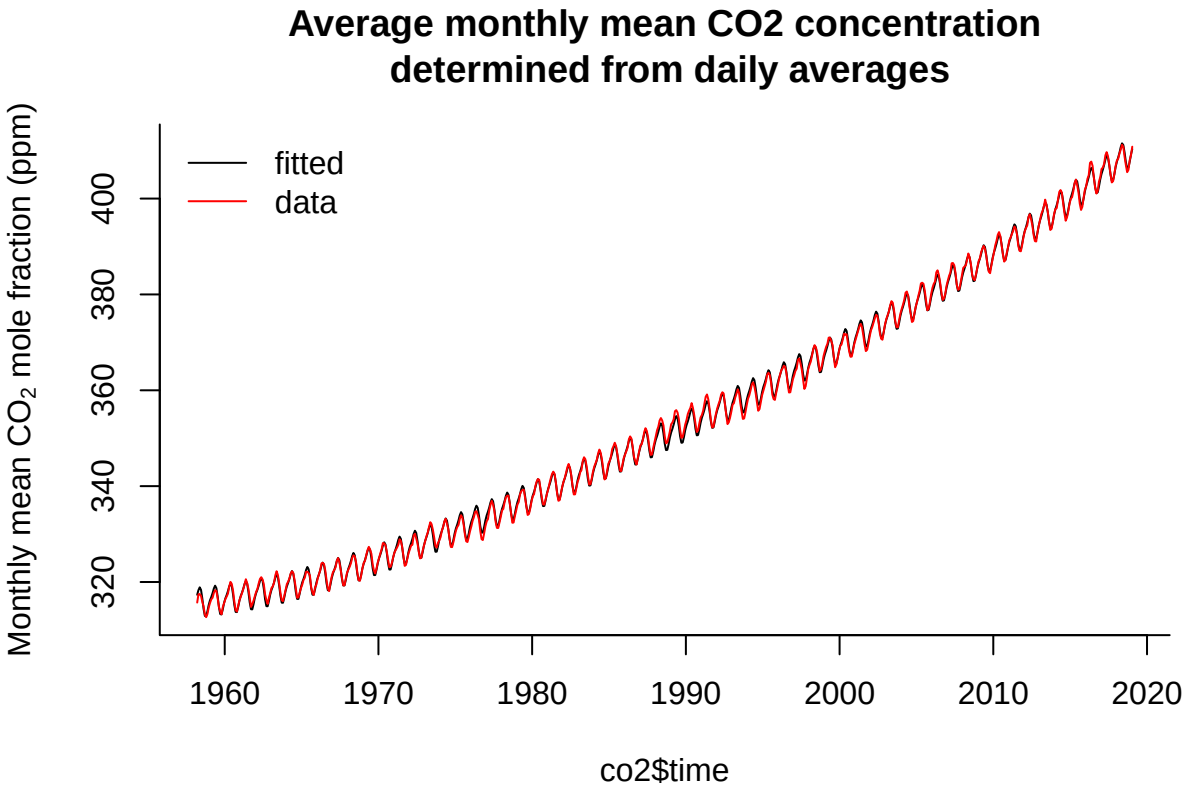

```
# Cast the full time series into a ts object
co2_ts <- with(co2, ts(data = interpolated, start = c(year[1], month[1]), end = c(tail(year,
    1), tail(month, 1)), frequency = 12))
with(co2, plot(jitter(month, 1/3), residuals(lin_mod), ylab = "Residuals", xlab = "Month of year",
    main = mcap))
```



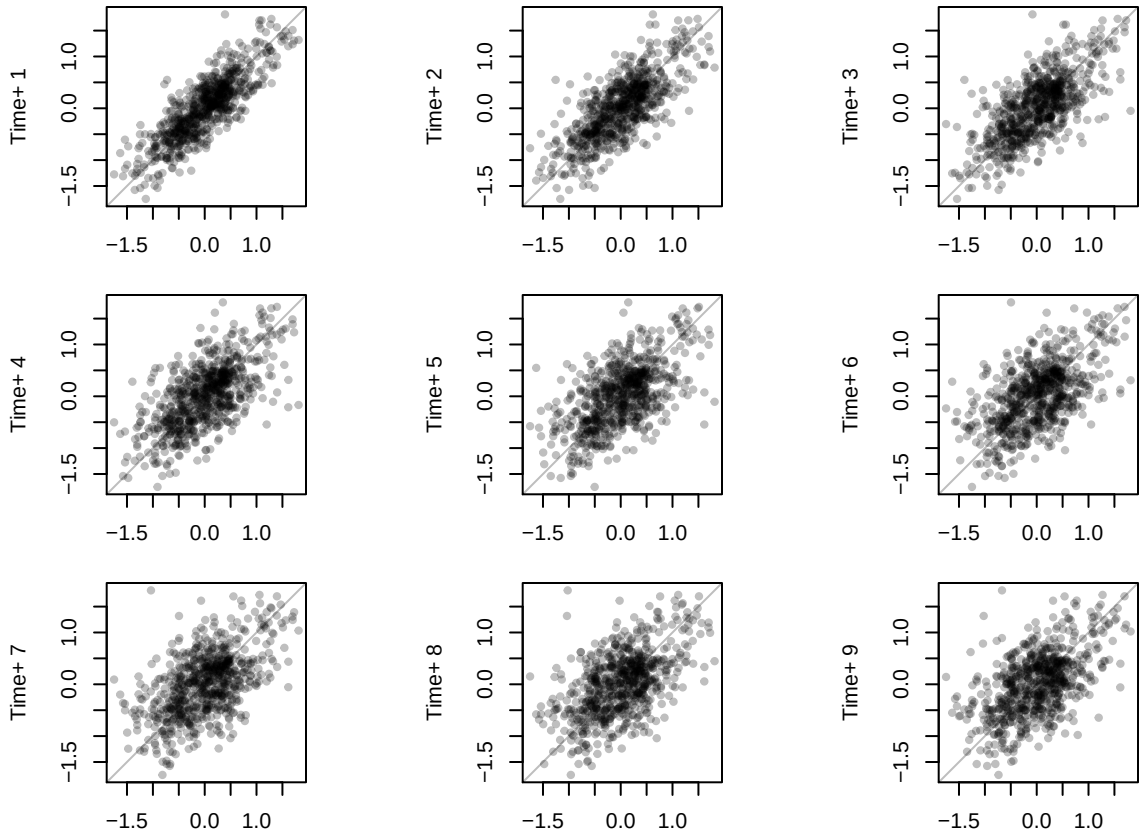

```
# Could create the basis manually
f_bs <- with(co2, fda::fourier(month, nbasis = 4, period = 12))[, -1]
lin_mod <- with(co2, lm(interpolated ~ splines::bs(time, df = 5, degree = 3) +
    f_bs))
# summary(lin_mod) Is there structure left in the residuals?
plot(co2$time, residuals(lin_mod), ylab = "Residuals from final `lm` model",
    xlab = "Time", main = mcap)
```



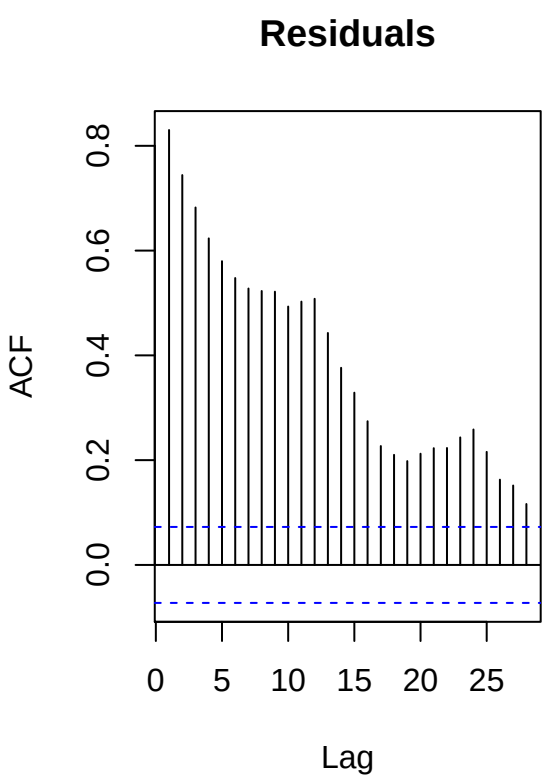

```
plot(co2$time, fitted(lin_mod), type = "l", ylab = ycap, main = mcap, bty = "l")
with(co2, lines(time, interpolated, col = 2))
legend(x = "topleft", legend = c("fitted", "data"), col = c(1, 2), lty = c(1,
    1), bty = "n")
```

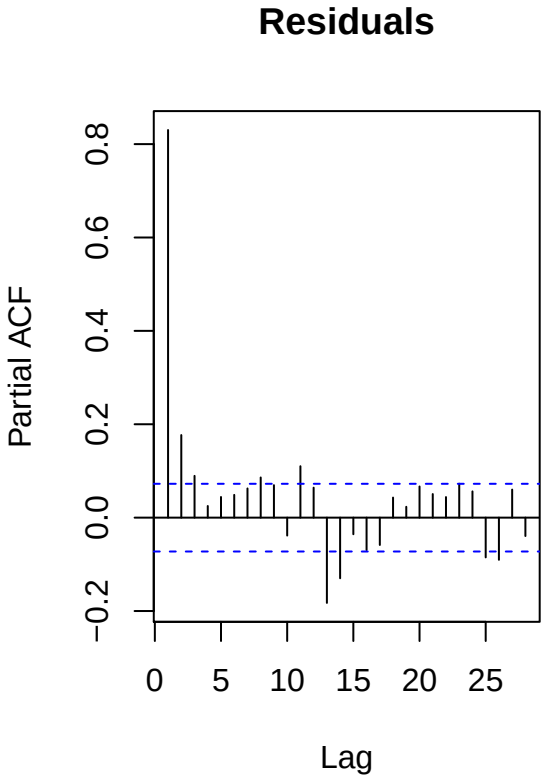



```r
# Trend not quite adequate because more exponential growth.  The trend does
# poorly in low-high observations Some discrepancy between the frequencies
# and the fitted Creates residual harmonic patterns - because trend minus
# fitted
res <- residuals(lin_mod)

pairs.ts <- function(d, lag.max = 10) {
    old_par <- par(no.readonly = TRUE)
    n <- length(d)
    X <- matrix(NA, n - lag.max, lag.max)
    col.names <- paste("Time+", 1:lag.max)
    for (i in 1:lag.max) X[, i] <- d[i - 1 + 1:(n - lag.max)]
    par(mfrow = c(3, 3), pty = "s", mar = c(3, 4, 0.5, 0.5))
    lims <- range(X)
    for (i in 2:lag.max) plot(X[, 1], X[, i], panel.first = {
        abline(0, 1, col = "grey")
    }, xlab = "Time", ylab = col.names[i - 1], xlim = lims, ylim = lims, pch = 20,
        col = rgb(0, 0, 0, 0.25))
    par(old_par)
}
pairs.ts(res)
```

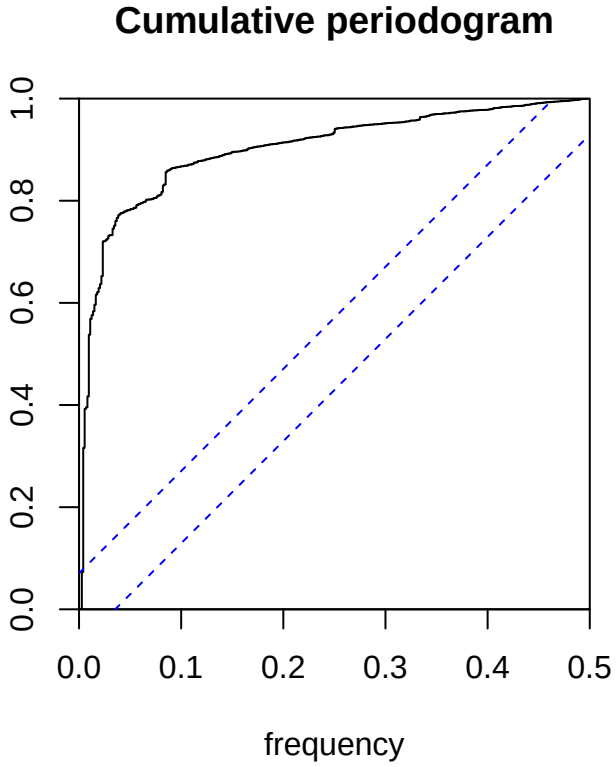



```
par(mfrow = c(1, 2))
TSA::acf(res, main = "Residuals")
pacf(res, main = "Residuals")
```

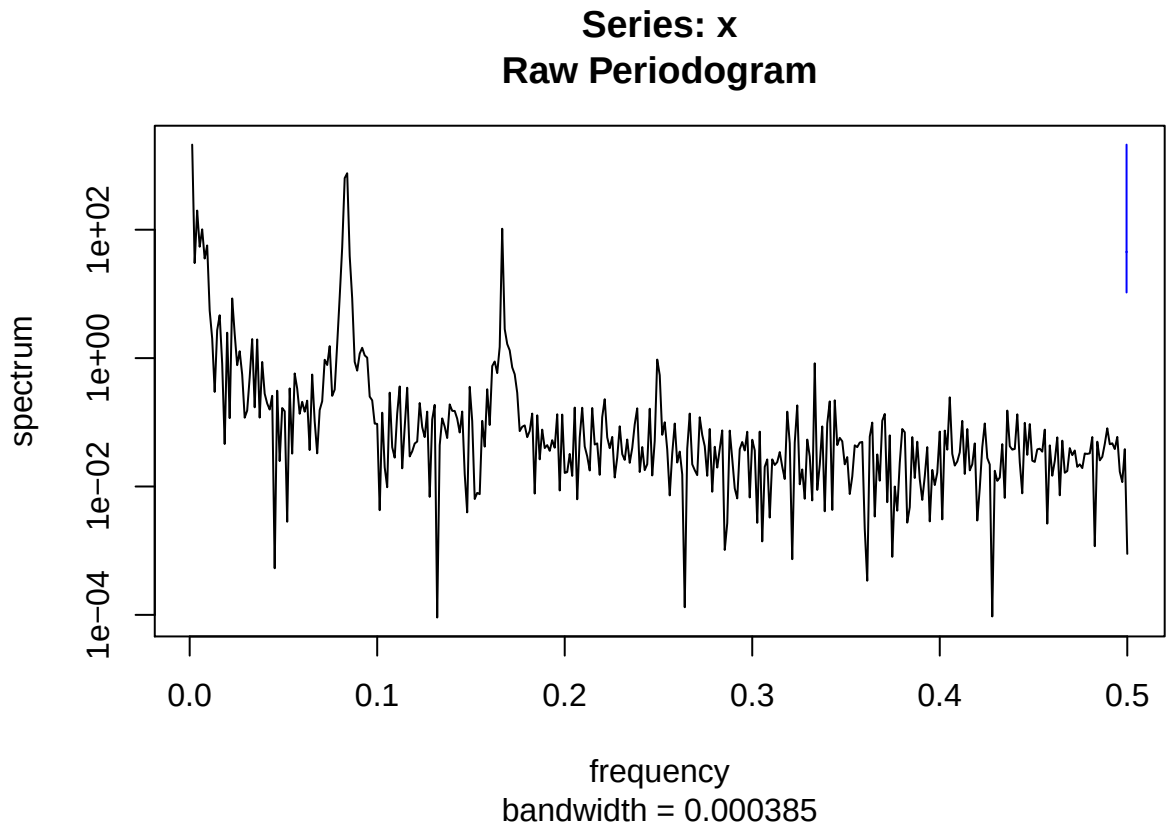




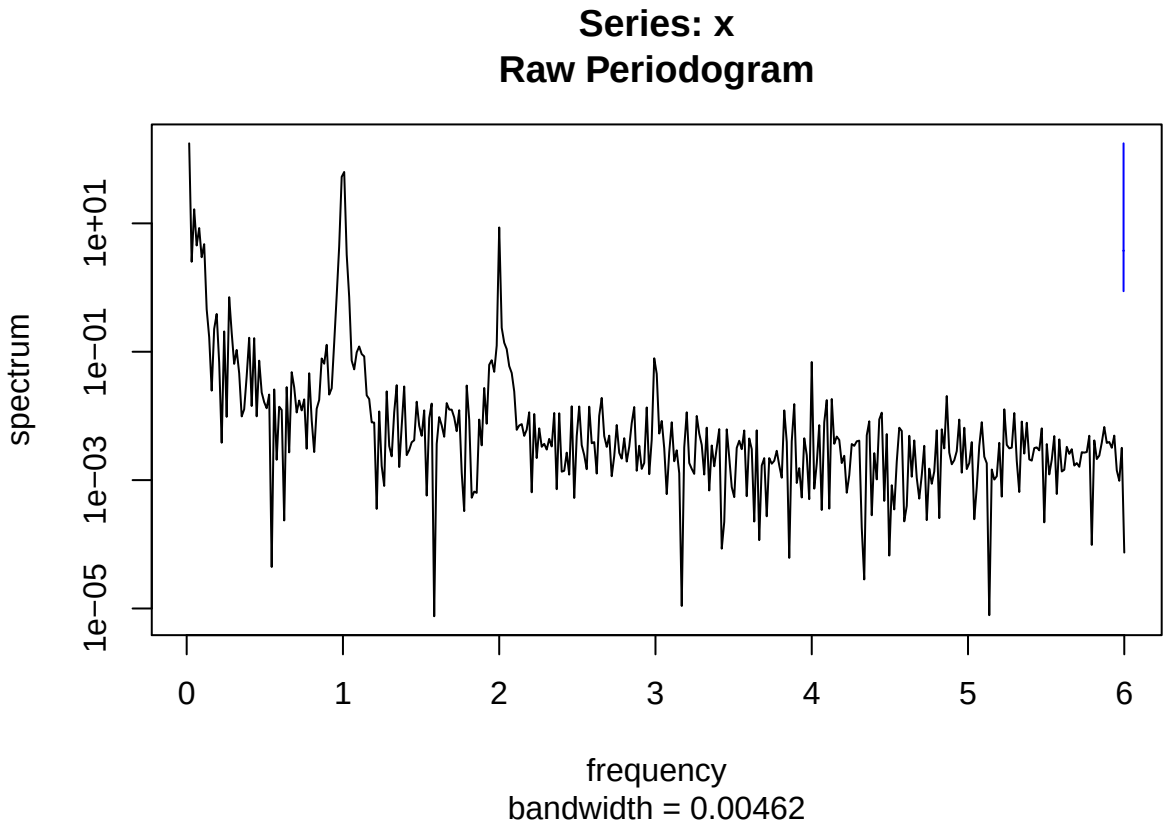

```
par(mfrow = c(1, 1))
# KS test: are residuals white noise?
cpgram(res, main = "Cumulative periodogram")
```



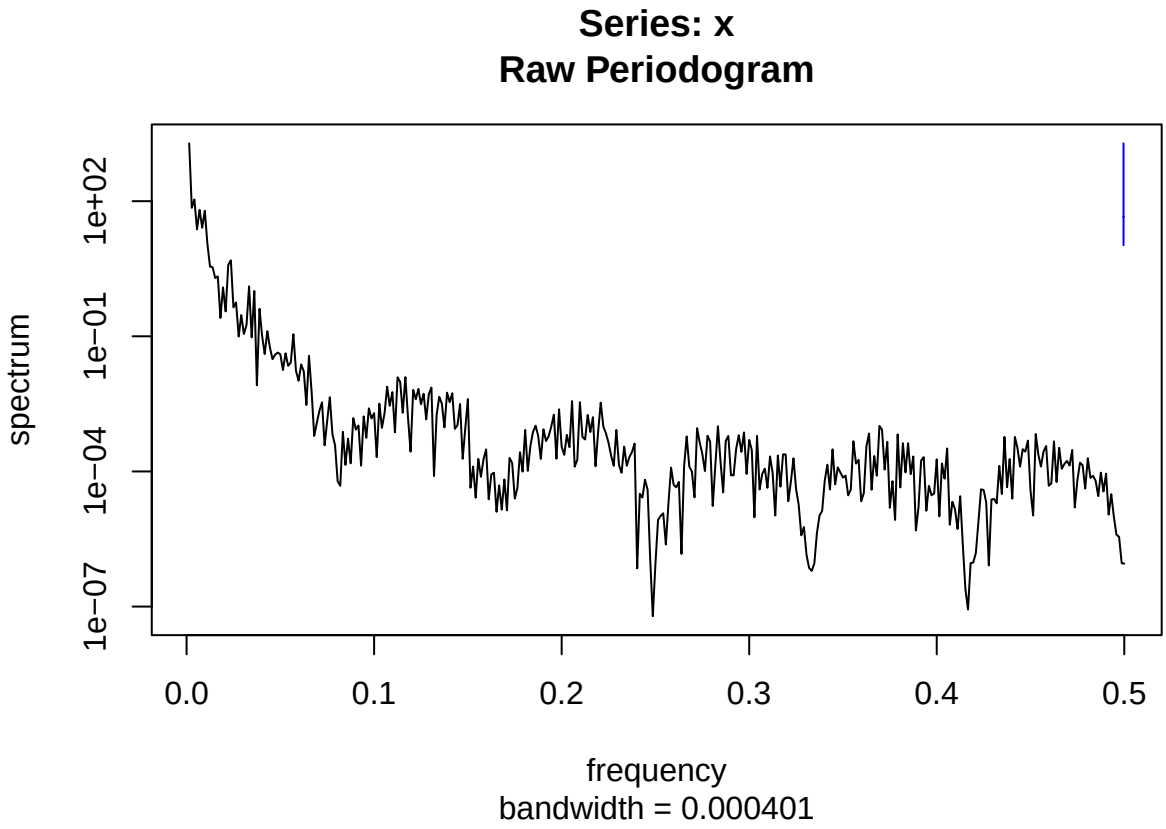
```
# No, as one would expect

## Spectrum of raw series
spectrum(co2$interpolated)
```



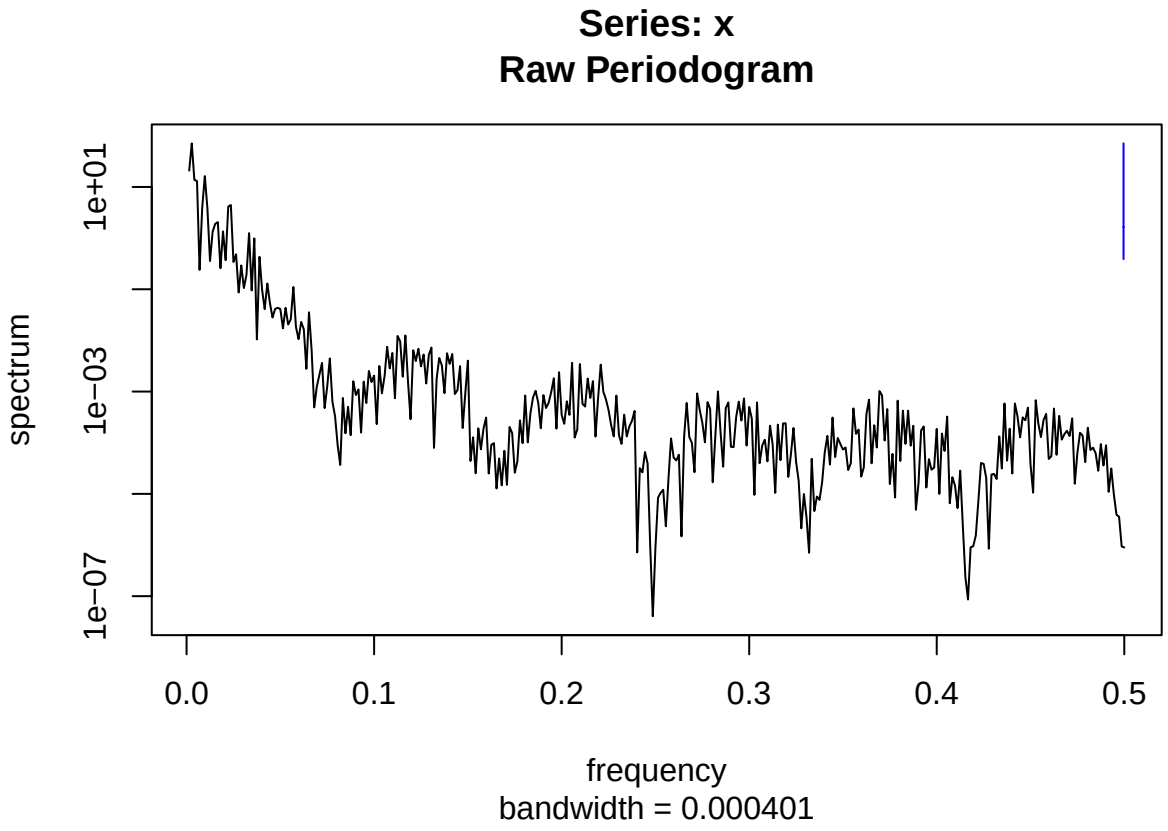

```
# default with vector is to have frequency on [0,0.5]
spectrum(co2_ts)  #otherwise corresponds to frequency of `ts`, here yearly
```




```
filtered <- filter(co2$interpolated, method = "convolution", filter = rep(1/12,
    12))
spectrum(na.contiguous(filtered))
```




```r
# Detrended smoothed series
spectrum(resid(lm(filtered ~ poly(co2$time, 2))))
```




```
# Test for H0
`?`(tseries::kpss.test)
tseries::kpss.test(res, null = "Level")
```

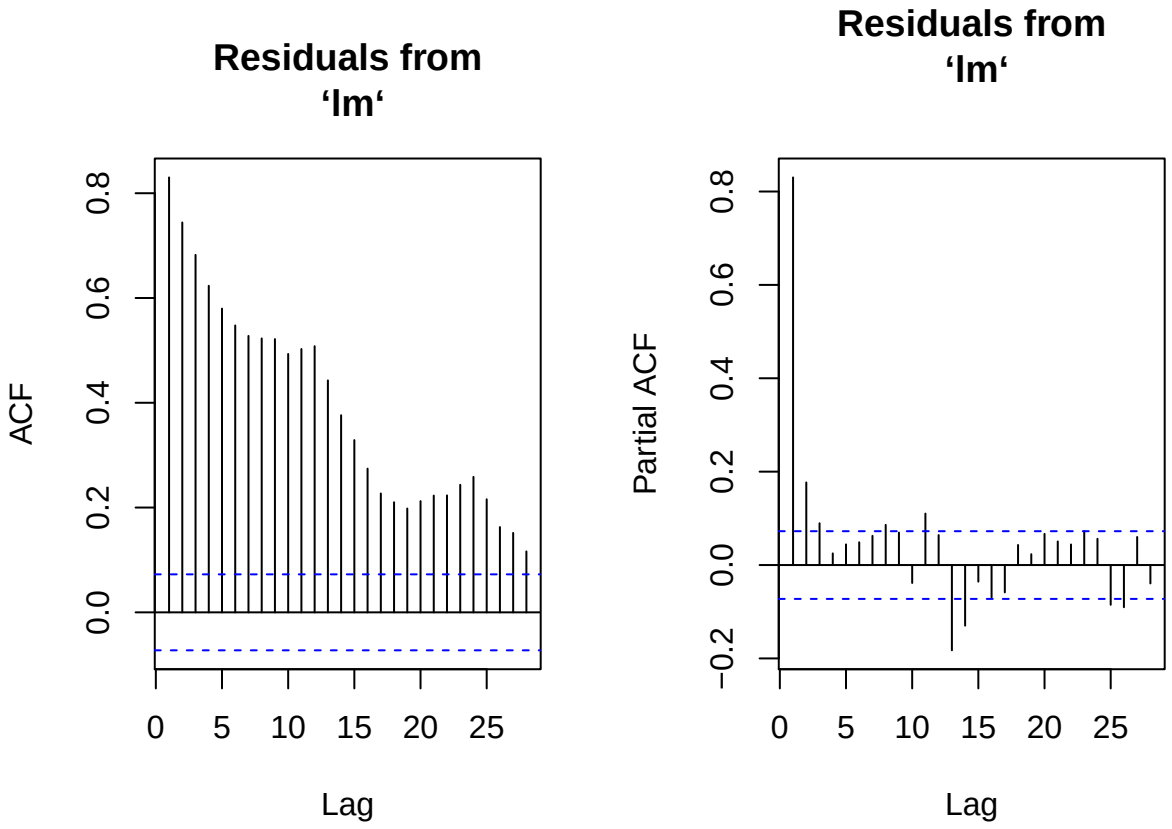
```
    KPSS Test for Level Stationarity

data:  res
KPSS Level = 0.16743, Truncation lag parameter = 6, p-value = 0.1
```

```
# Fail to reject null that it is level stationary
tseries::kpss.test(res, null = "Trend")
```

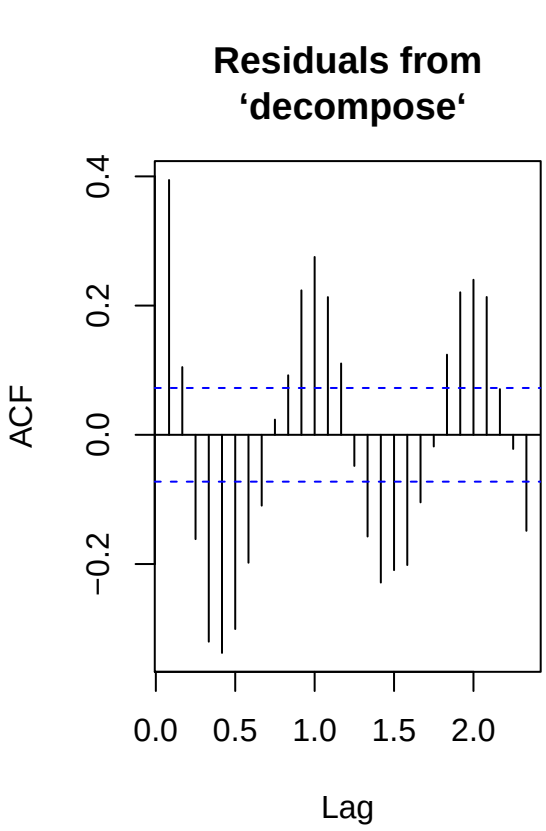
```
    KPSS Test for Trend Stationarity

data:  res
KPSS Trend = 0.16743, Truncation lag parameter = 6, p-value =
0.03214
```

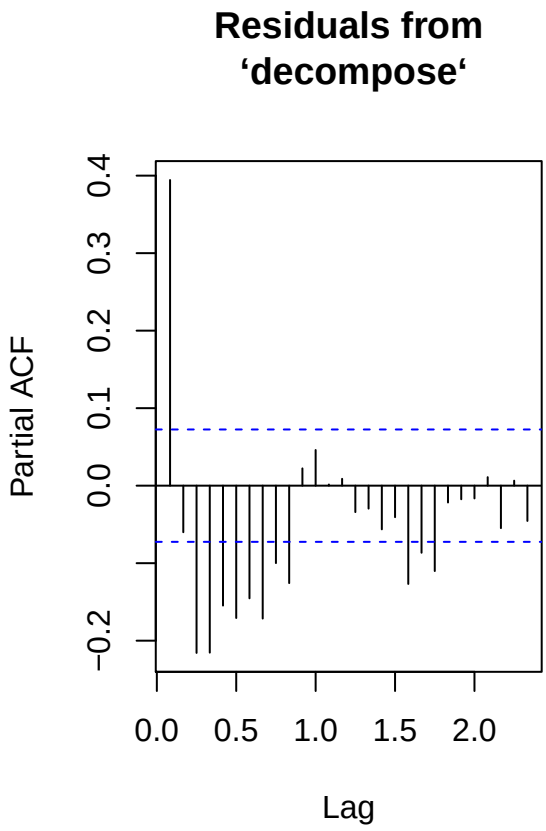
```
# Reject null at 5% that it is trend stationary

res_dec <- decompose(co2_ts)$random
res_stl <- stl(co2_ts, s.window = "periodic")$time.series[, "remainder"]

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

```
# Some structure left due to incorrect model specification Residual
# frequency at lag 12-24 and two lag residuals
TSA::acf(res, na.action = na.pass, main = "Residuals from\n `lm`")
pacf(res, na.action = na.pass, main = "Residuals from\n `lm`")
```

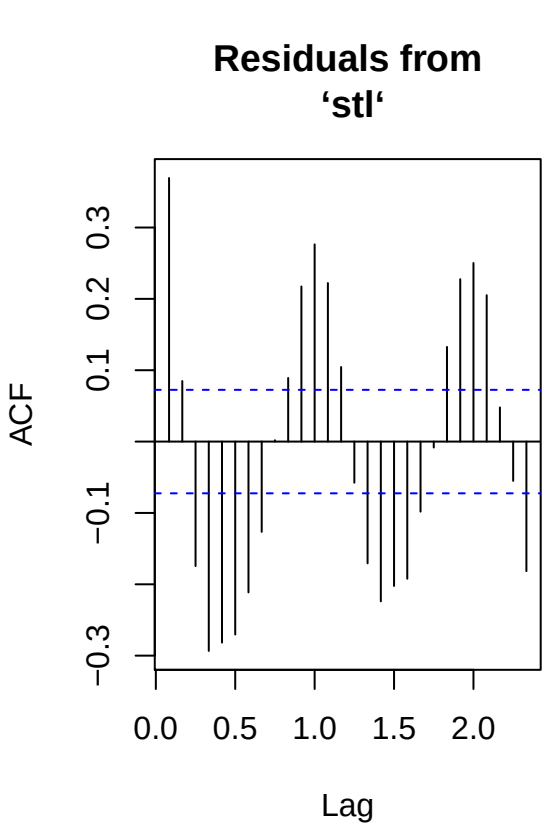


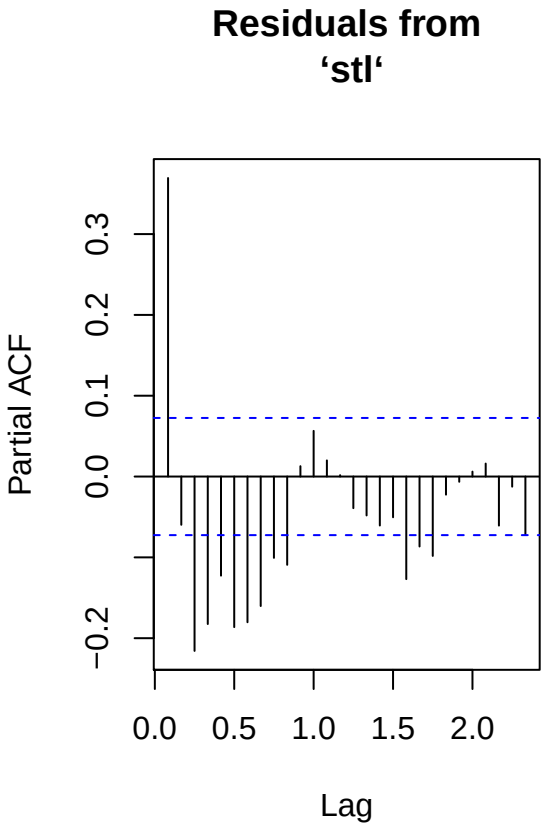

```
# Residuals show some remaining periodicity at year 1. Would need AR(1)
# model
TSA::acf(res_dec, na.action = na.pass, main = "Residuals from\n `decompose`")
pacf(res_dec, na.action = na.pass, main = "Residuals from\n `decompose`")
```






```r
# Similar output
TSA::acf(res_stl, main = "Residuals from\n `stl`")
pacf(res_stl, main = "Residuals from\n `stl`")
```


Residuals from
'stl'
ACF
0.3
0.2
0.1
−0.1
−0.3
0.0
0.5
1.0
1.5
2.0
Lag


Residuals from
'stl'
Partial ACF
0.3
0.2
0.1
0.0
−0.2
0.0
0.5
1.0
1.5
2.0
Lag

Chapter 2

Likelihood estimation and the Box–Jenkins method

This tutorial addresses the following:

- estimation of ARIMA and ARCH models using conditional maximum likelihood.
- the Box–Jenkins methodology for selecting the order of ARMA processes based on analysis of the (partial) correlogram.
- model selection using information criterion, unit roots and problems arising from model fit.

2.1 Manual maximum likelihood estimation

As was done in class for the `beaver` dataset, we will look at manual specification of the likelihood. While it is straightforward in principle to maximize the latter for ARMA models, the numerous restrictions that are imposed on the parameters make it hard, if not impossible, to manually code one's own function. Maximum likelihood estimation is implemented typically via the state-space representation, which we will cover later in the semester.

For simple models, it is easily done however, and should shed some light on the various functions that are part of **R** for optimization, the definition of a function, the use of `nlm` and `optim` for optimization purposes, etc.

We first load a dataset of UBS and Credit Suisse stock prices from 2000 until 2008. The data is splitted in three parts for the analysis, since the data is heteroscedastic, and there appears (visually) to be two changepoints. We look at the adequacy of fitted AR(1) model for the mean and an ARCH(1) for the variance.

```
# devtools::install_github('nickpoison/astsa')
# devtools::install_github('joshuaulrich/xts')
library(xts)
library(lubridate)
# read data and examine it
UBSCreditSuisse <- read.csv("http://sma.epfl.ch/~lbelzile/math342/UBSCSG.csv",
    stringsAsFactors = FALSE)
names(UBSCreditSuisse)
```

```
 [1] "Date"       "UBS_OPEN"   "UBS_HIGH"   "UBS_LOW"    "UBS_LAST"
 [6] "UBS_VOLUME" "CSG_OPEN"   "CSG_HIGH"   "CSG_LOW"    "CSG_LAST"
[11] "CSG_VOLUME"
```

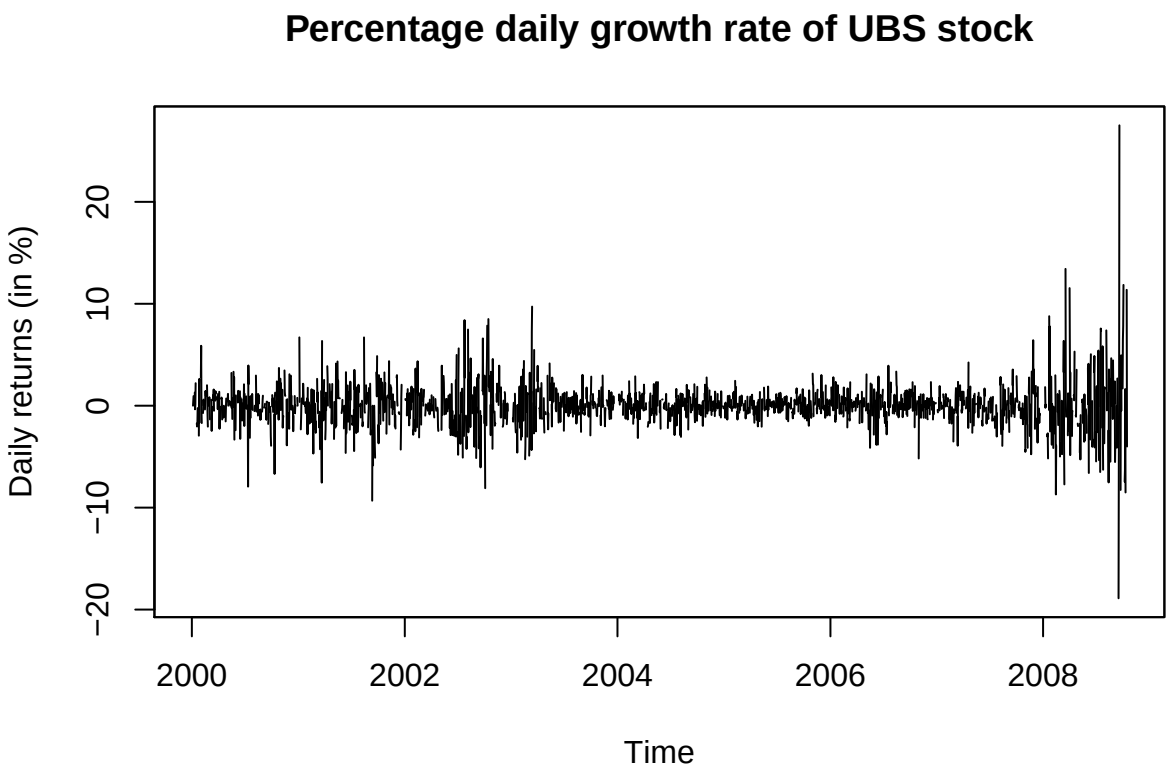
```
head(UBSCreditSuisse)
```

```
    Date UBS_OPEN UBS_HIGH UBS_LOW UBS_LAST UBS_VOLUME CSG_OPEN CSG_HIGH
1 1/1/00       NA       NA      NA       NA          NA       NA       NA
2 1/2/00       NA       NA      NA       NA          NA       NA       NA
3 1/3/00       NA       NA      NA       NA          NA       NA       NA
4 1/4/00    31.13    31.17   30.32    30.32    11526322    72.01    72.13
5 1/5/00    30.06    31.06   29.73    30.32    17142124    67.51    69.01
6 1/6/00    30.29    30.80   30.25    30.47     9509228    68.09    68.55
  CSG_LOW CSG_LAST CSG_VOLUME
1      NA       NA         NA
2      NA       NA         NA
3      NA       NA         NA
4   69.13    69.24    5336924
5   67.40    68.44    4419160
6   67.74    68.55    2585800
```

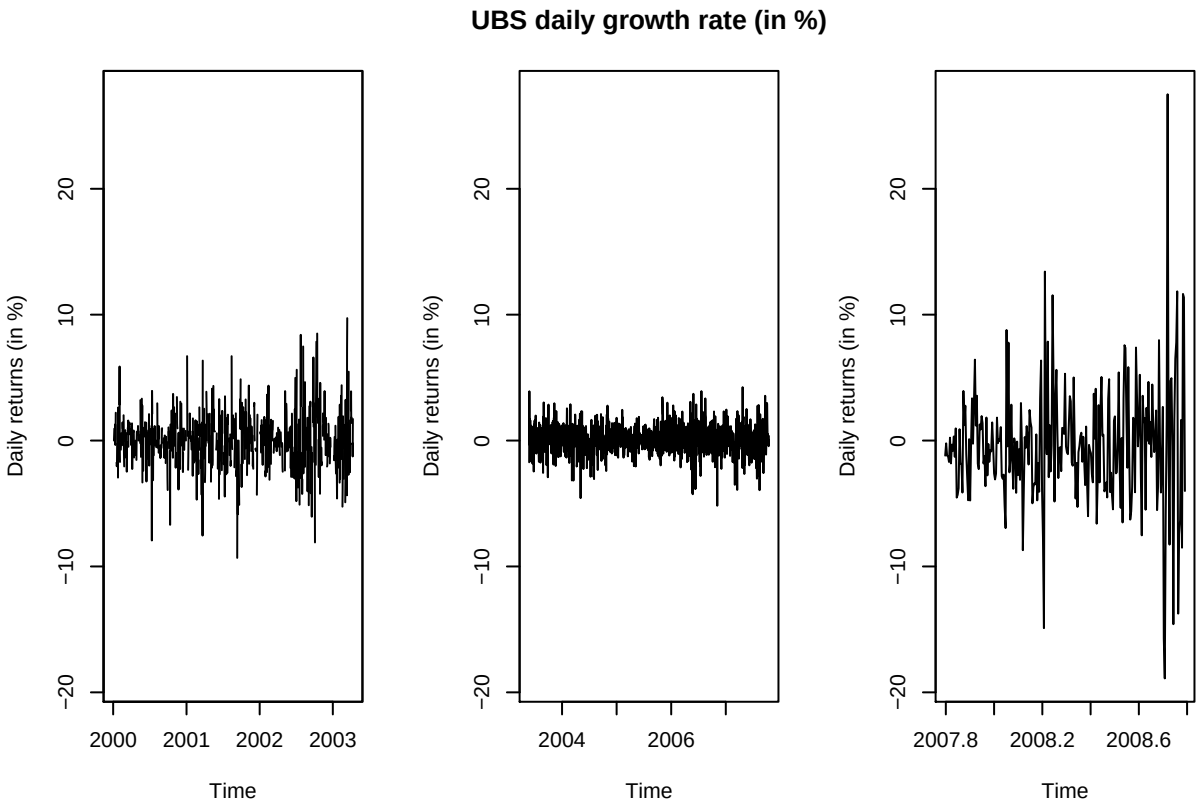
```
# create time series, accounting for missing values at weekends and 251.25
# values/year this is correct for analysis, but only provides approximate
# locations for plotting
UBS <- ts(UBSCreditSuisse$UBS_LAST, start = c(2000, 1), frequency = 365.25)
UBS <- ts(UBS[!is.na(UBS)], start = c(2000, 1), frequency = 251.625)

# Irregular time series
UBS_xts <- with(UBSCreditSuisse, xts(UBS_LAST, mdy(Date)))
```

Objects of class `ts` store the dates from the vector start with observations as $(i-1)/\omega$. Thus, we specified in the above a vector encoded as 2000, 2000+1/365.25, ... This means that missing values are not handled. In contrast, `xts` objects keep the time stamps from a `Date` object. The function `with` is equivalent to `attach`, but has a limited scope and is used to avoid writing `UBSCreditSuisse$Date`, etc. The function `mdy` transforms the string `Date` as month, day and year. The string is coerced into an object of class `Date`.

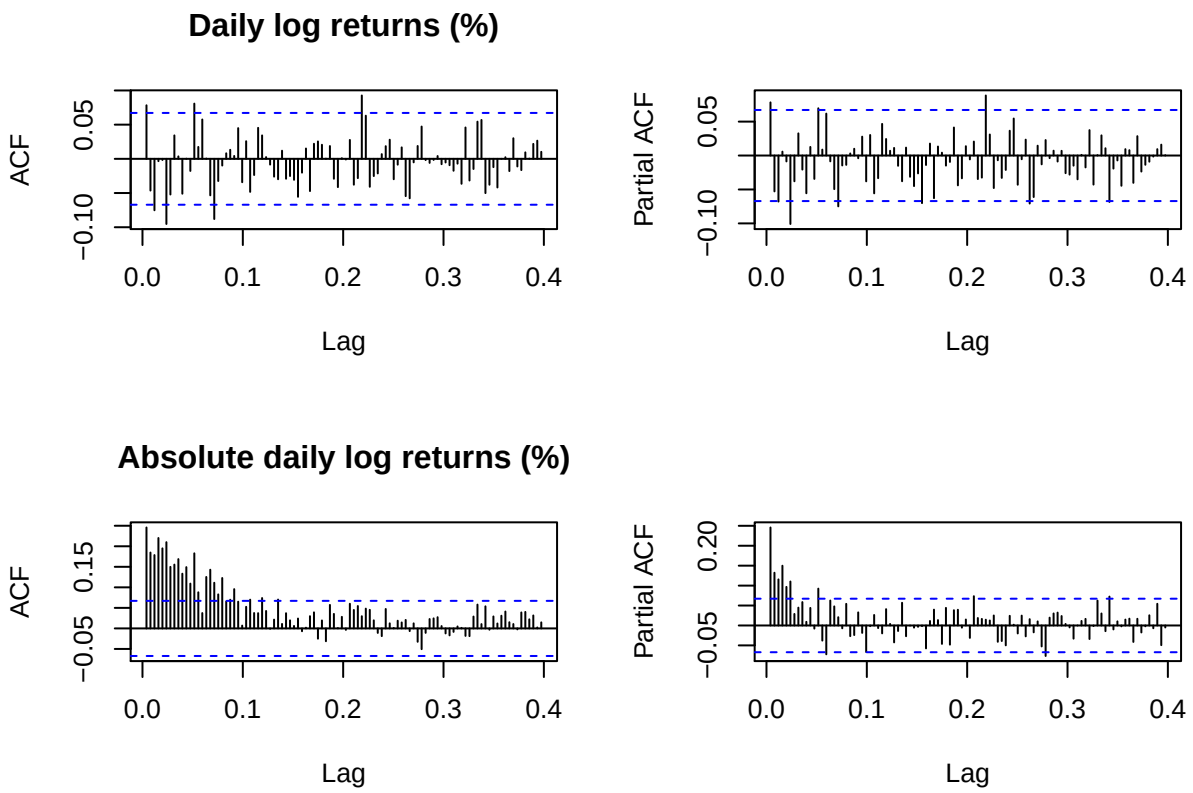
```
# Analysis for UBS returns, 2000-2008
UBS_ret <- 100 * diff(log(UBS_xts))
plot.zoo(UBS_ret, xlab = "Time", ylab = (ylab <- "Daily returns (in %)"), main = "Percentage daily grow
```




```
# compute log returns
UBS.ret <- 100 * diff(log(UBS))

# split into 3 homogeneous(?) parts, and plot using the same vertical axis
# on the graphs

# with the xts object
par(mfrow = c(1, 3))
lims <- range(UBS.ret)
plot.zoo(UBS_ret[paste0(index(first(UBS_ret)), "/", as.Date("2003-01-01") +
    100)], ylim = lims, xlab = "Time", ylab = ylab)

# with window and the ts object
y1 <- window(UBS.ret, end = c(2003, 100))
# plot(y1, ylim = lims)

y2 <- window(UBS.ret, start = c(2003, 101), end = c(2007, 200))
plot(y2, ylim = lims, ylab = ylab, main = "UBS daily growth rate (in %)")

y3 <- window(UBS.ret, start = c(2007, 201))
plot(y3, ylim = lims, ylab = ylab)
```




```
# analysis of first part, first just plotting ACF and PACF for data and for
# abs(data)
y <- y1

# (Partial) correlograms for the series
par(mfrow = c(2, 2))
TSA::acf(y, lag.max = 100, main = "Daily log returns (%)")
pacf(y, lag.max = 100, , main = "")
TSA::acf(abs(y), lag.max = 100, main = "Absolute daily log returns (%)")
pacf(abs(y), lag.max = 100, main = "")
```



The residuals look pretty much white noise, but the variance has residual structure. Recall the implicit definition of the AR(1) process Y_t,

$$Y_t = \mu + \phi(Y_{t-1} - \mu) + \varepsilon_t,$$

where $\varepsilon_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$. The joint distribution of the observations conditional on the first is multivariate normal. Here is a simple function for the likelihood, which only requires specifying the conditional mean.

```
# analysis using AR(1) model for means conditional likelihood
nll_AR1 <- function(th, y) {
    n <- length(y)
    condit.mean <- th[1] + th[3] * (y[-n] - th[1])
    -sum(dnorm(y[-1], mean = condit.mean, sd = th[2], log = TRUE))
}
init1 <- c(0, 1, 0.5)
# fit1 <- nlm(f = nll_AR1, p = init1, iterlim = 500, hessian = TRUE, y = y)
fit1 <- optim(init1, nll_AR1, y = y, hessian = TRUE, method = "Nelder-Mead")
```

We obtain the parameter estimates and the standard errors from the observed information matrix, estimated numerically. Incidently, one can easily that the problem is equivalent to a linear Gaussian model where the regressor is a lagged vector of observations. The parameter estimates differ slightly, but this is due to the optimization routine.

```
#Parameter values (MLEs)
fit1$par
#Standard errors from inverse of Hessian matrix at MLE
#If you code the optimization routine yourself, you can still obtain the Hessian via
#hessian <- numDeriv::hessian(func = nll_AR1, y = y, x = fit1$par)

#Standard errors
```

```
sqrt(diag(solve(fit1$hessian)))

#Conditional likelihood using lm
#dynlm is a wrapper around lm for `ts` and `zoo` objects, L means lag and you can add e.g. trend(y)
fit1_ols <- dynlm::dynlm(y ~ L(y, 1))
coefficients(fit1_ols)
sd(residuals(fit1_ols))
```

Incidently, the situation is analogous for the ARCH(1) process, which has a conditional variance that changes over time. The latter is defined implicitly as

$$Z_t = \mu + \sigma_t \epsilon_t$$
$$\sigma_t^2 = \alpha_0 + \alpha_1 (Z_{t-1} - \mu)^2$$

with $\epsilon_t \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$.

The variance σ^2 here is included as $\sigma^2 = \alpha_0$ and the parameter appearing in the likelihood is α_1/σ^2 corresponds to θ_3, or `th[3]`.

```
# analysis using ARCH(1) model for variances
nll_ARCH1 <- function(th, y) {
    n <- length(y)
    condit.mean <- th[1]
    condit.var <- th[2] * (1 + th[3] * (y[-n] - th[1])^2)
    -sum(dnorm(y[-1], mean = condit.mean, sd = sqrt(condit.var), log = TRUE))
}
init2 <- c(0, 1, 0.5)
fit2 <- nlm(f = nll_ARCH1, p = init2, iterlim = 500, hessian = TRUE, y = y)
## fit2 <- optim(init2, nll_ARCH1, y = y, hessian = TRUE)
```

The function `nlm` performs minimization, but may return warnings because some of its steps because the conditional variance can be negative for some combinations of the variable, so the corresponding moves of the Newton algorithm are rejected. These are typically steps that are not in the neighborhood of the final solution, so can be ignored if the output is valid. The `minimum` corresponds to the negative log-likelihood at the maximum likelihood estimates. The MLE is given by `estimate` and the standard errors by the square root of the diagonal entries of the inverse Hessian (here already negated because we work with the negative of the log-likelihood). Since the residuals have a varying variance, we need to adjust them by dividing each by their respective variance. Same would have occured for the AR(1) process, but it is easier for the mean.

```
fit2$minimum
```

```
[1] 1860.386
```

```
fit2$estimate
```

```
[1] 0.02691498 3.31180313 0.11770094
```

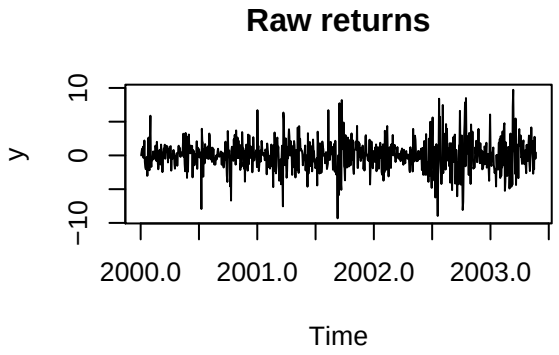
```
sqrt(diag(solve(fit2$hessian)))
```

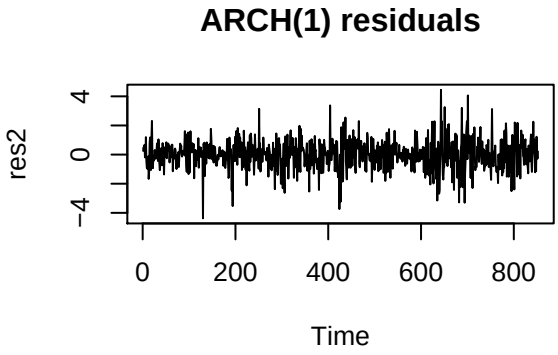
```
[1] 0.06843004 0.24664944 0.02908274
```

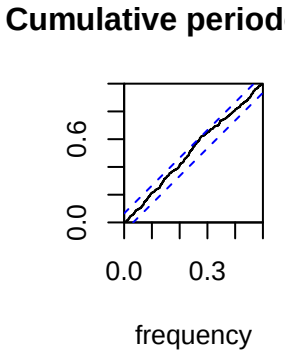
```
make_resid_ARCH1 <- function(y, fit) {
    th <- fit$estimate
    n <- length(y)
    condit.mean <- th[1]
    condit.var <- th[2] * (1 + th[3] * (y[-n] - th[1])^2)
    res <- (y[-1] - condit.mean)/sqrt(condit.var)
    ts(res)
}

res2 <- make_resid_ARCH1(y, fit2)
```

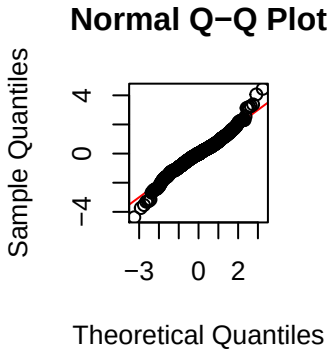
We now proceed with diagnostic plots to check the model adequacy. Recall that the Kolmogorov–Smirnov test statistic associated with the cumulative periodogram tests the hypothesis of white noise (and ARCH(1) is white noise).

```
par(mfrow = c(2, 2))
plot(y, main = "Raw returns")
plot(res2, main = "ARCH(1) residuals")
cpgram(res2, main = "Cumulative periodogram")
par(pty = "s")
qqnorm(res2, panel.first = {
    abline(0, 1, col = "red")
})
```









2.1.1 Exercise 1: UBS stock returns

1. Create a function that fits an AR(1)-ARCH(1) model by modifying the code provided above and apply it to `y`. The latter is defined as

$$Y_t - \mu = \phi(Y_{t-1} - \mu) + \sigma_t \varepsilon_t, \quad \sigma_t^2 = \alpha_0 + \alpha_1 (Y_{t-1} - \mu)^2, \quad \varepsilon_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$$

2. Obtain the maximum likelihood estimates using `nlm` or `optim` as well as the standard errors
3. Plot the residuals. Comment on the fit using standard diagnostic plots (Q-Q plot, ((P)ACF, cumulative periodogram).
4. Fit an AR(2) model using a conditional likelihood for the mean and obtain the standard errors of your estimated coefficients.
5. Perform a likelihood ratio test to test whether the AR(2) coefficient is significative.

2.2 Box–Jenkins methodology for ARMA models

The Wold decomposition theorem states that any second-order stationary time series can be represented as a deterministic process and a stochastic linear process, which can be represented as a causal MA(∞) series of the form

$$Y_t = \sum_{j=0}^{\infty} \psi_j \varepsilon_{t-j}, \qquad \varepsilon_t \sim \text{WN}(0, \sigma^2), \quad \psi_0 = 1, \quad \sum_{j=0}^{\infty} \psi_j^2 < \infty$$

This does provide a justification for using an ARMA model for modelling. The latter may not be parsimonious nor useful at characterizing the data generating mechanism, but they nevertheless provide in some simple cases a good approximation.

The Box–Jenkins methodology for ARMA models (dating back to time where computing ressources were scarce) allows one to select the order of an AR(p), MA(q) or ARMA(p, q) by visual inspection of the (partial) correlograms. Both should always go alongside one another.

1. Apply a transformation of the data X_t where appropriate

- logarithm, Box–Cox transform
- differencing so that the series appears linear.

2. Correlogram

- Determine the MA(q) order by looking at the autocorrelation, at the points for which $\rho_k \neq 0$ for $k \leq q$ and $r_k \approx 0$ for $k > q$.
- For an AR(p) process, the autocorrelation function should decay exponentially, with possible oscillation patterns.
- For an ARMA(p, q) model, the pattern is irregular for lags $k = 1, \ldots, p$ and go to zero as $k \to \infty$.

3. Partial correlogram

- Parameters should be zero at lags $k > p$ for the AR(p) model, and nonzero otherwise
- The parameters decay exponentially in the MA(q) model
- The parameters decrease to zero as $k \to \infty$ for the ARMA(p, q) model.

The function to fit these models is `arima`, whose arguments are specified via `order = c(p, d, q)`. Ignore the `d` component for now in the triple (p, d, q) by setting it to zero. Other options are `sarima` from `astsa`, which is a wrapper around `arima`. `sarima` provides diagnostic plots alongside and includes a constant by default, but the syntax differs from `arima` and it takes directly components p, d and q. The function `Arima` from Hyndman's `forecast` package is yet another wrapper around `arima`. An explanation of the differences can be found in Forecasting: principles and practice[1], at the bottom of the page.

[1] https://www.otexts.org/fpp/8/7

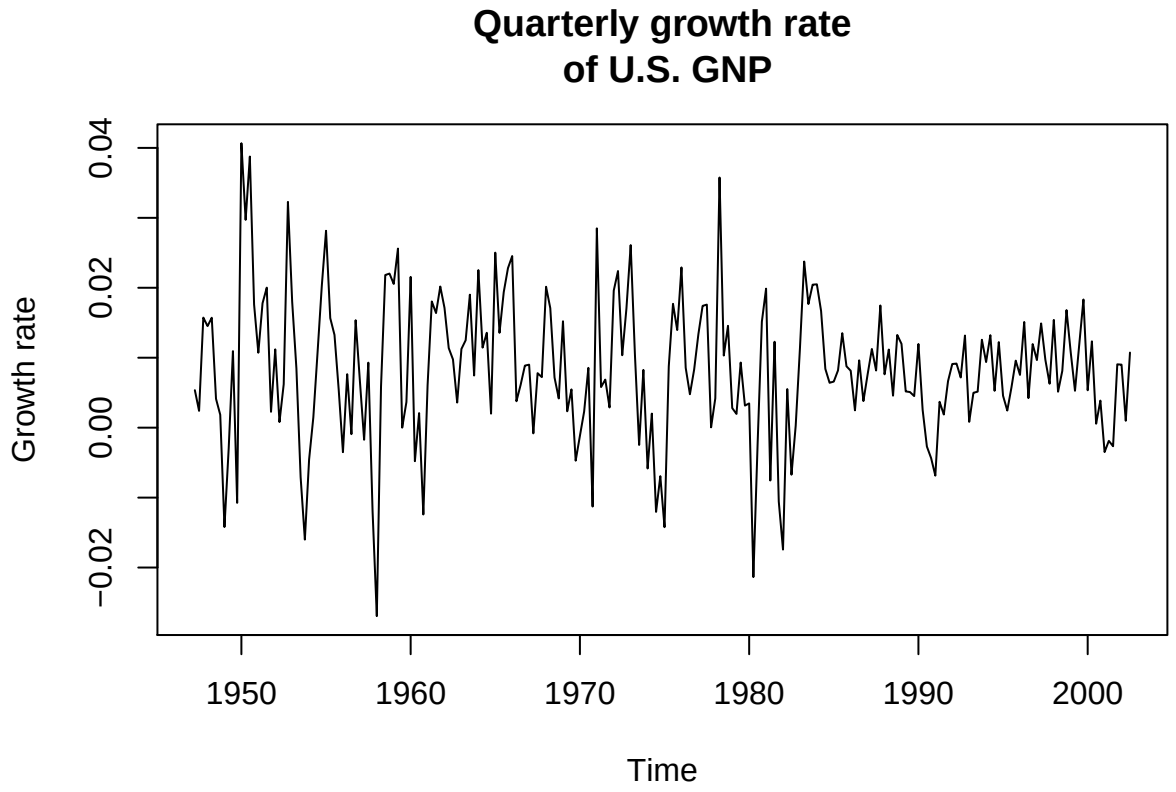
The `Arima()` command from the `forecast` package provides more flexibility on the inclusion of a constant

It also correctly labels the latter. Depending on your model, it may be the level (mean), an intercept, the linear trend (slope, or drift in the time serie literature). If we take first difference, the constant is the drift, etc.

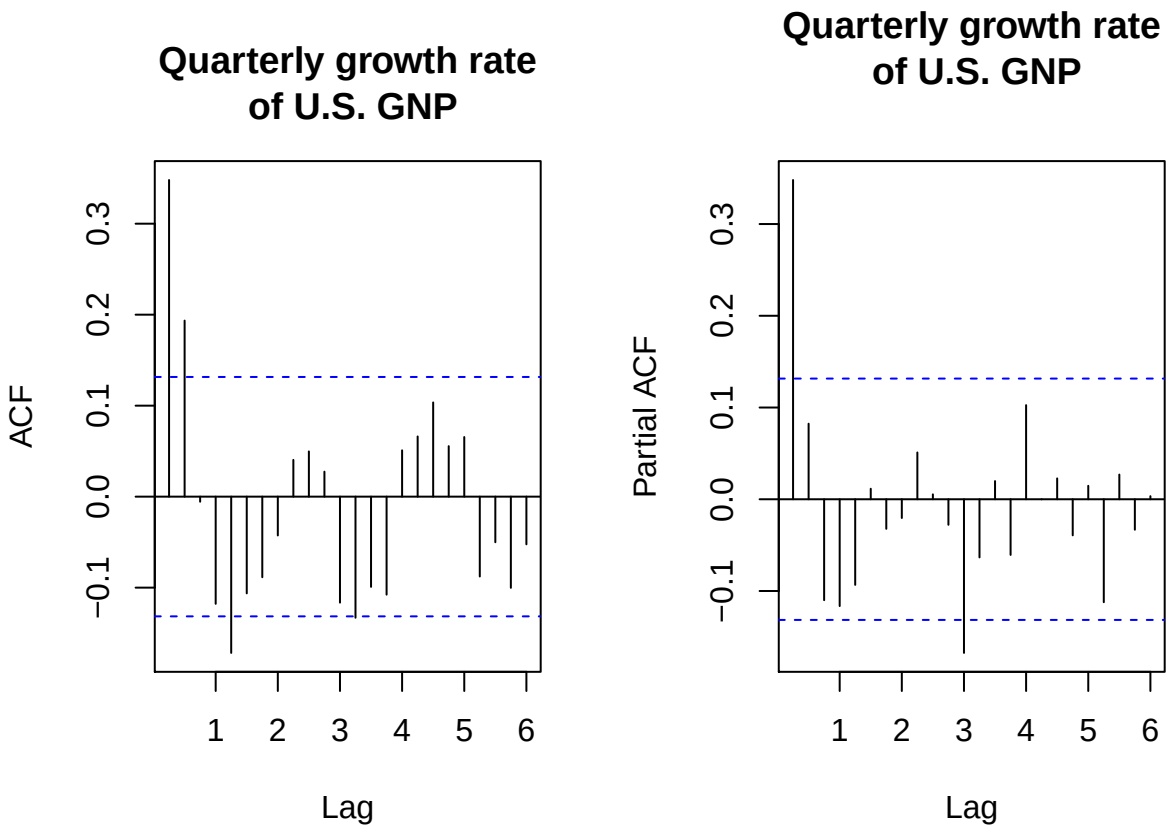
Warning: You may stumble on the web on `auto.arima`. Beware of the naive and automated selection implemented by this function (which relies on what I would consider to be an *ad hoc* forward model selection). Use at your own risk.

ARMA models can be almost equivalent, as the following example from Shumway and Stoffer (example 3.28) illustrates. Note that as we use `sarima`, a constant is included by default. We can assess its significance by the usual t-test, if the error structure is appropriate.

```
library(astsa)
gnpgr = diff(log(gnp))  # growth rate of GNP
main <- "Quarterly growth rate\n of U.S. GNP"
plot(gnpgr, main = main, ylab = "Growth rate")
```




```
# There is a different mean in each quarter, but forego the seasonal effect
# This is obvious in the following plot, which plots (in order), separating
# by quarter monthplot(gnpgr, main = main, ylab = 'Growth rate', xlab =
# 'Quarter')
par(mfrow = c(1, 2))
TSA::acf(gnpgr, 24, main = main)
pacf(gnpgr, 24, main = main)
```




```
# What does the period in the correlogram correspond to?
```

The decrease in ACF/PACF suggests that either an AR(1) or an MA(2) might be appropriate here.

```
mod1 <- sarima(gnpgr, p = 1, d = 0, q = 0, details = FALSE)  # AR(1)
print(mod1$ttable)
```

```
      Estimate     SE t.value p.value
ar1     0.3467 0.0627  5.5255       0
xmean   0.0083 0.0010  8.5398       0
```

```
mod2 <- sarima(gnpgr, p = 0, d = 0, q = 2, details = FALSE)  # MA(2)
print(mod2$ttable)
```

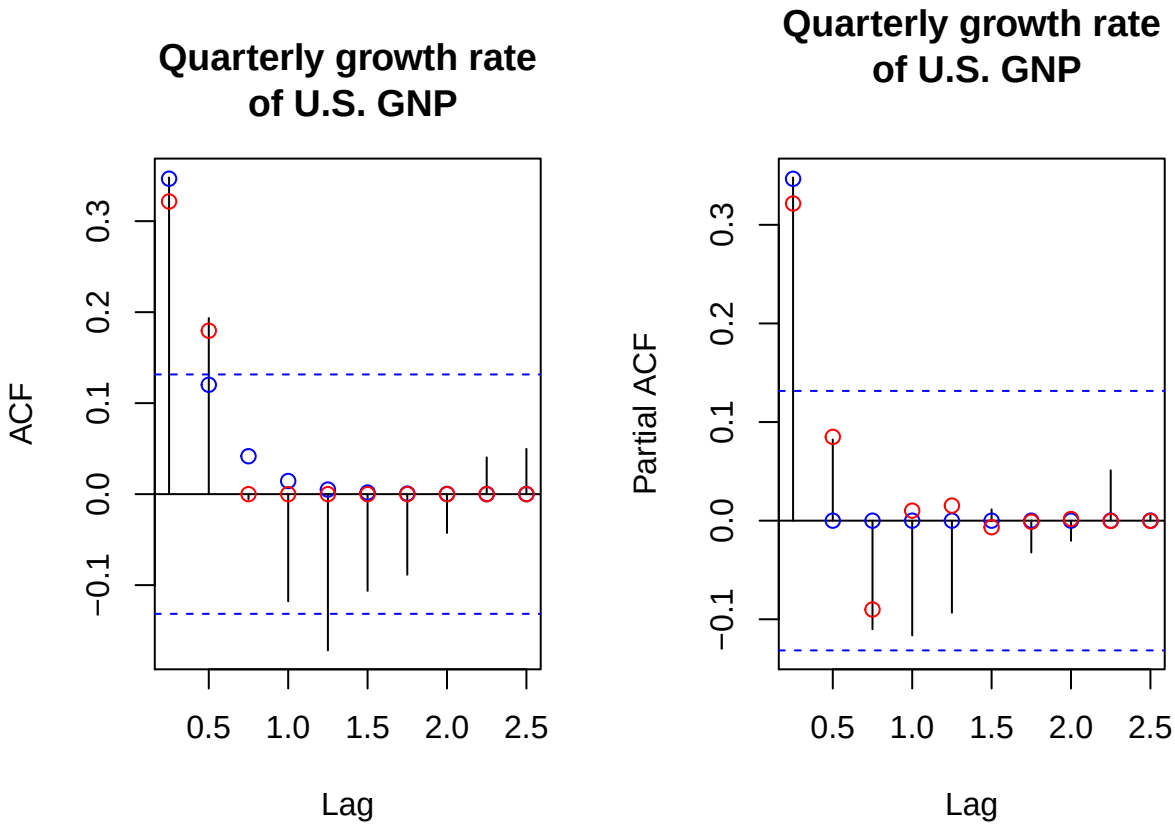
```
      Estimate     SE t.value p.value
ma1     0.3028 0.0654  4.6272  0.0000
ma2     0.2035 0.0644  3.1594  0.0018
xmean   0.0083 0.0010  8.7178  0.0000
```

Having fitted these two models, we compare them using their linear proces (or MA(∞) representation) and the theoretical autocorrelation and partial autocorrelation coefficients.

```
# Obtain the coefficients from the MA(Inf)
ARMAtoMA(ar = mod1$fit$coef[1], ma = 0, 4)  # prints first 4 psi-weights
```

```
[1] 0.34665450 0.12016934 0.04165724 0.01444067
```

```r
# Print sample ACF and superpose the theoretical coefficients implied by the
# model
par(mfrow = c(1, 2))
TSA::acf(gnpgr, 10, main = main)
points(seq(0, by = 0.25, length = 11), y = ARMAacf(ar = mod1$fit$coef[1], lag.max = 10),
    col = "blue")
points(seq(0, by = 0.25, length = 11), y = ARMAacf(ma = mod2$fit$coef[1:2],
    lag.max = 10), col = "red")

pacf(gnpgr, 10, main = main)
points(seq(0.25, by = 0.25, length = 10), y = ARMAacf(ar = mod1$fit$coef[1],
    lag.max = 10, pacf = TRUE), col = "blue")
points(seq(0.25, by = 0.25, length = 10), y = ARMAacf(ma = mod2$fit$coef[1:2],
    lag.max = 10, pacf = TRUE), col = "red")
```



Warning: in the example above, it is necessary here to add a constant coefficient. Economic theory suggest exponential growth, hence a trend of $\exp(t\beta)$. This becomes a linear trend $t\beta$ for log returns, and by differencing, we obtain the trend β. The latter corresponds to the long term trend, or the quarterly growth rate, which is about 1%. The coefficient is significant even if it seems small and failing to include it leads to invalid inference about the state of the U.S. economy.

The last step is to check whether our process is causal and invertible. For the former, `arima` typically transforms to the stationary and invertible solution, so we should be good. We may end up having a root of $\Phi(B)$ or $\Theta(B)$ on the unit circle. In this case, the ARMA process is not stationary, while in the second, asymptotic normality of the estimator breaks down. We look at the roots of the polynomial $\Phi(B)$ and $\Theta(B)$ and check they indeed are outside the unit circle. It is easiest to check their norm or modulus.

```
# Is the MA(2) process invertible?
abs(polyroot(c(1, mod2$fit$coef[1:2])))  #MA roots
```

```
[1] 2.216653 2.216653
```

```
# Is the AR(1) process causal
Mod(polyroot(c(1, -mod1$fit$coef[1])))  #AR roots
```

```
[1] 2.884717
```

Sometimes models give similar output. The model choice should then be made on the grounds of **parsimony**: a simpler model that explains the data should be prefered. If the models are nested, for example an AR(1) and an AR(2), then a likelihood ratio test can be performed (watch out to make sure the same number of observations appear in the models if using a conditional likelihood). A quick way to do this is by looking at the Akaike's information criterion. There are other information criterion, notably the BIC and bias-corrected version of AIC termed AIC_c. They are defined respectively as

$$\begin{aligned}\text{AIC} &= -2\ell(\boldsymbol{\theta}) + 2k,\\ \text{BIC} &= -2\ell(\boldsymbol{\theta}) + k\log(n),\\ \text{AIC}_\text{c} &= -2\ell(\boldsymbol{\theta}) + 2\frac{kn}{n-k-1}.\end{aligned}$$

In the formulas above, n is the sample size, k is the number of parameters in the model and ℓ is log-likelihood. Since likelihood values increase with the number of parameters provided the models are nested (why?), the additional term penalizes complex models. The Schwartz's information criterion (or Bayesian information criterion, BIC in short) includes the sample size in the penalty. The penalty could be viewed as a testing procedure similar to likelihood ratio test for nested models, where in place of the quantiles of the χ^2 distribution, one uses e.g. for AIC a threshold of $2(k_1 - k_2)$.

The BIC is consistent, meaning that it chooses the correct model with probability 1 as $n \to \infty$. AIC is *not* consistent for model selection and this results typically in overfitting. For ARMA models models with zero mean, Brockwell and Davis advocate the use of a small-sample correction of AIC, AIC_c, with $k = p + q + 1$. The latter is equivalent to AIC when $n \to \infty$.

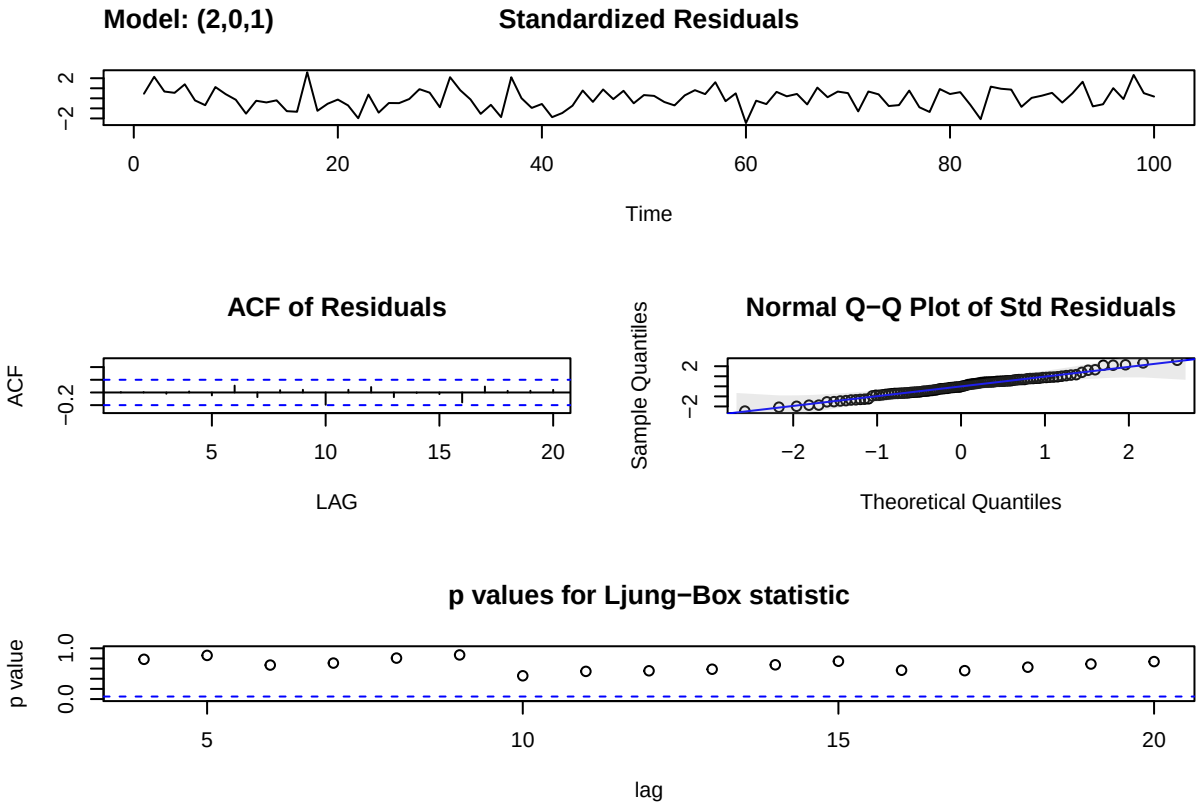
Warning: The information criterion returned by `astsa::sarima` does not correspond to the definition above; the one implemented can be found in Shumway and Stoffer's book. The principle remains the same: the lower, the better.

Here is an illustrative example on the dataset in the following exercise with a helper function to find the roots of the polynomials. I would like to bring your attention to the fact about the difficulties of optimizing some time series models. For example, one often finds that the ARMA(2,1) fits as well as the AR(1), even when the latter is the true underlying model. However, a quick look at the output below is illustrative of the trouble that lies ahead.

In the next exercise, you will be asked to practice your skills on simulated datasets. Before letting you proceed, I want to illustrate the different commands for fitting e.g. an ARMA(2, 1) model, and see what comes out.

```
library(forecast)
library(astsa)
load(url("http://sma.epfl.ch/~lbelzile/math342/Simulated_ARMA.RData"))

# Wrapper around arima from forecast library
mod_a <- arima(x3, order = c(2, 0, 1), transform.pars = TRUE)
mod_b <- Arima(x3, order = c(2, 0, 1), transform.pars = TRUE)
```

```
# Arima returns BIC, AIC and AICC
mod_c <- sarima(x3, p = 2, d = 0, q = 1, details = FALSE)
```




```
# sarima provides diagnostic plots unless `details=FALSE` unfortunately also
# returns all the optimization steps from `arima` in the new version
mod_a
```

```
Call:
arima(x = x3, order = c(2, 0, 1), transform.pars = TRUE)

Coefficients:
         ar1      ar2     ma1  intercept
      1.0872  -0.3739  0.9621    -0.6908
s.e.  0.1142   0.1182  0.1310     0.5769

sigma^2 estimated as 0.7271:  log likelihood = -128.77,  aic = 267.54
```

```
mod_b
```

```
Series: x3
ARIMA(2,0,1) with non-zero mean

Coefficients:
         ar1      ar2     ma1     mean
      1.0872  -0.3739  0.9621  -0.6908
s.e.  0.1142   0.1182  0.1310   0.5769
```

```
sigma^2 estimated as 0.7574:  log likelihood=-128.77
AIC=267.54   AICc=268.18   BIC=280.57
```

```
mod_c
```

```
$fit

Call:
stats::arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D,
    Q), period = S), xreg = xmean, include.mean = FALSE, optim.control = list(trace = trc,
    REPORT = 1, reltol = tol))

Coefficients:
         ar1      ar2     ma1    xmean
      1.0872  -0.3739  0.9621  -0.6908
s.e.  0.1142   0.1182  0.1310   0.5769

sigma^2 estimated as 0.7271:  log likelihood = -128.77,  aic = 267.54

$degrees_of_freedom
[1] 96

$ttable
      Estimate     SE t.value p.value
ar1     1.0872 0.1142  9.5220  0.0000
ar2    -0.3739 0.1182 -3.1627  0.0021
ma1     0.9621 0.1310  7.3417  0.0000
xmean  -0.6908 0.5769 -1.1973  0.2341

$AIC
[1] 0.7612783

$AICc
[1] 0.7876612

$BIC
[1] -0.1345149
```

You can (and should) check the roots of the AR and MA polynomials to make sure this is not happening. The argument `transform.pars` transforms the AR polynomial to the causal representation if it does not have a root on the unit circle. If the process is not invertible, the parameter lies on the boundary of the space and the usual standard errors obtained from the observed information matrix are not reliable. It is possible to use these models for forecasting, but their interpretation is awkward.

2.2.1 Exercise 2: Simulated series

1. The `Simulated_ARMA` dataset contains 8 time series (these examples are from Charlotte Wickham's website). Fit ARMA models to those and select a model, justifying your choice. Be skeptical of the optimization routine and analyze the models carefully.

2.3 Information criterion, model selection and profile likelihood

The following example is due to Edward Ionides[2] and is licensed under CC-BY-NC.

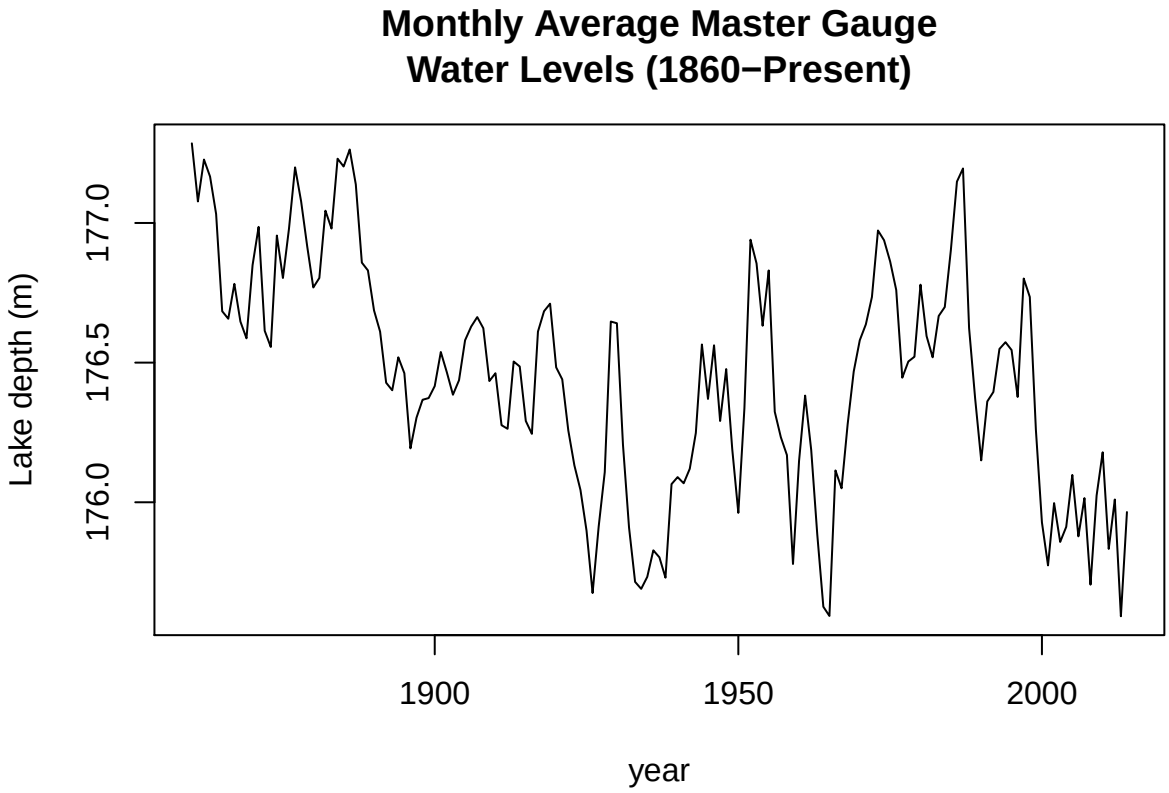
We analyze data from NOAA, the depth of Lake Huron (in m). There is a decreasing trend. We first format the time variable and extract the height in January.

```
library(lubridate)
# Data from
# http://www.glerl.noaa.gov/data/dashboard/data/levels/mGauge/miHuronMog.csv
dat <- read.csv("http://sma.epfl.ch/~lbelzile/math342/huron_depth.csv", sep = ",",
    header = TRUE, skip = 2)
dat$Date <- strptime(dat$Date, "%m/%d/%Y")
dat$year <- with(dat, year(Date))
dat$month <- with(dat, month(Date))  #as.numeric(format(dat$Date, format = '%m'))
head(dat)
```

```
        Date Average year month
1 1860-01-01 177.285 1860     1
2 1860-02-01 177.339 1860     2
3 1860-03-01 177.349 1860     3
4 1860-04-01 177.388 1860     4
5 1860-05-01 177.425 1860     5
6 1860-06-01 177.461 1860     6
```

```
## Subsample values to take only January data
dat <- subset(dat, month == 1)
huron_depth <- dat$Average
year <- dat$year
# Plot the series
plot(huron_depth ~ year, type = "l", ylab = "Lake depth (m)", main = "Monthly Average Master Gauge\nWat
```

[2] http://ionides.github.io/531w16/notes05/notes5.html



Next, we create a table containing the AIC values, which are obtained from fitting successive ARMA models with varying orders p, q. Here, a simple AR(1) does an excellent job at capturing the structure and all other models are essentially more complicated versions.

```
huron_ar1 <- Arima(huron_depth, order = c(1, 0, 0), include.drift = TRUE, include.mean = TRUE)

## Table of AIC values extracted from the output - notice the warnings on
## your computer when the fit fails!
aic_table <- function(data, P, Q) {
    table <- matrix(NA, (P + 1), (Q + 1))
    for (p in 0:P) {
        for (q in 0:Q) {
            table[p + 1, q + 1] <- arima(data, order = c(p, 0, q))$aic
        }
    }
    dimnames(table) <- list(paste("<b> AR", 0:P, "</b>", sep = ""), paste("MA",
        0:Q, sep = ""))
    table
}
huron_aic_table <- aic_table(huron_depth, 4, 5)
require(knitr)
kable(huron_aic_table, digits = 2)
```

	MA0	MA1	MA2	MA3	MA4	MA5
AR0	166.75	46.60	7.28	-14.97	-18.64	-26.09
AR1	-38.00	-37.41	-35.46	-33.82	-34.13	-32.20
AR2	-37.33	-38.43	-36.90	-34.93	-34.35	-33.08
AR3	-35.52	-35.17	-32.71	-31.38	-31.13	-32.98
AR4	-33.94	-34.91	-34.43	-36.27	-31.31	-30.90

```
## Kables are tables for HTML. For LaTeX, use something like library(xtable)
## dimnames(huron_aic_table) <- list(paste0('AR', 0:4), paste0('MA', 0:5))
## latex_tab <- xtable::xtable(huron_aic_table, booktabs = TRUE, caption =
## 'AIC values for ARMA models for the Lake Huron dataset') print(latex_tab,
## booktabs = TRUE, caption.placement = 'top')
```

I silenced the output, but some of the optimization routine *failed* (duh). You can see this by looking at the AIC values (recall the definition and look at nested models). These cannot increase by more than 2 from left to right or top to bottom.

Subsequent quotes are from Prof. Ionides' notes

> What do we learn by interpreting the results in the above table of AIC values? In what ways might we have to be careful not to over-interpret the results of this table?

We can look at the roots to see if the process of our choice (the one with lowest AIC, say) is good. Overly complicated models can lead you to big troubles. We illustrate this by fitting a complex model, an ARMA(2,1), to the series:

```
## Fit an ARMA(2, 1) model
huron_arma21 <- Arima(huron_depth, order = c(2, 0, 1), include.drift = TRUE,
    include.mean = TRUE)
huron_arma21
```

```
Series: huron_depth
ARIMA(2,0,1) with drift

Coefficients:
          ar1     ar2     ma1  intercept    drift
      -0.0979  0.7464  1.0000   176.8391  -0.0049
s.e.   0.0553  0.0554  0.0244     0.1734   0.0019

sigma^2 estimated as 0.04188:  log likelihood=26.89
AIC=-41.78   AICc=-41.22   BIC=-23.52
```

```
## Root of the Phi polynomial
AR_roots <- polyroot(c(1, -coef(huron_arma21)[grep("^ar", names(huron_arma21$coef))]))
Mod(AR_roots)
```

```
[1] 1.224905 1.093792
```

The process is causal, but the estimate of the MA coefficient, θ_1 is on the boundary of the parameter space and numerically indistinguishable from 1.

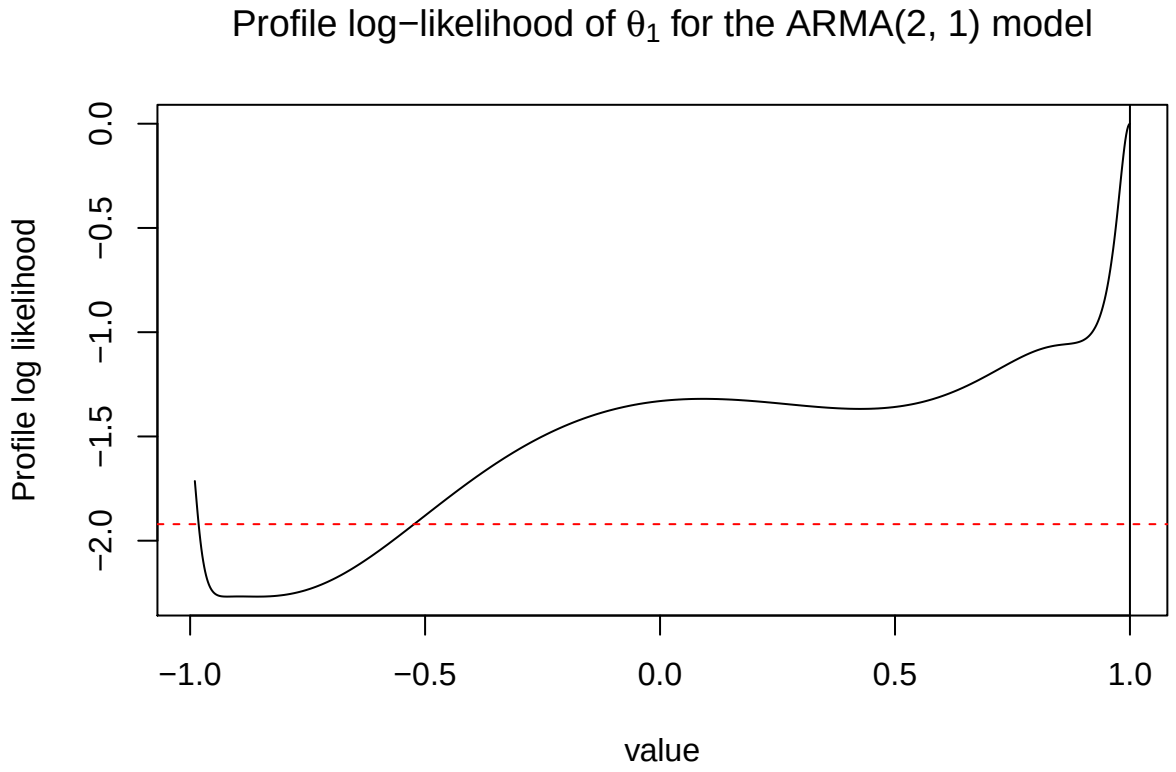
> Let's investigate a little, using profile methods. The claimed standard error on the MA(1) coefficient, from the Fisher information approach used by `arima` is small. We can see if the approximate confidence interval constructed using profile likelihood is in agreement with the approximate confidence interval constructed using the observed Fisher information.
>
> Note that `arima` transforms the model to invertibility. Thus, the estimated value of θ_1 can only fall in the interval $(1,1)$ but can be arbitrarily close to 1 or 1.

Recall the definition of profile likelihood: for a parameter of interest $\boldsymbol{\psi}$ and a partition $\boldsymbol{\theta} = (\boldsymbol{\psi}, \boldsymbol{\lambda})$, the profile likelihood as a function of $\boldsymbol{\psi}$ is

$$\ell_p(\boldsymbol{\psi}) = \text{argmax}_{\boldsymbol{\lambda}\in\Lambda} \ell(\boldsymbol{\lambda} \mid \boldsymbol{\psi}).$$

The maximum profile likelihood coincides with the maximum likelihood estimate (why?), and we use it normally to obtain more accurate confidence intervals for (a) parameter(s) of interest. Here, we investigate the profile log-likelihood of the MA(1) coefficient. Unlike the nice examples you saw in class, the profile here is weird and the global maximum lies on the boundary.

```
## Profile log-likelihood for the MA coefficient
K <- 1000
ma1 <- seq(from = -0.99, to = 1, length = K)
profile_loglik <- rep(NA, K)
for (k in 1:K) {
    profile_loglik[k] <- logLik(arima(huron_depth, order = c(2, 0, 1), xreg = scale(seq(1:length(huron_
        include.mean = TRUE, fixed = c(NA, NA, ma1[k], NA, NA)))
}
# Failed for some values of the MA
plot(profile_loglik - logLik(huron_arma21) ~ ma1, type = "l", ylab = "Profile log likelihood",
    xlab = "value", main = expression(paste("Profile log-likelihood of ", theta[1],
        " for the ARMA(2, 1) model")))
abline(h = -qchisq(0.95, 1)/2, col = 2, lty = 2)
abline(v = 1, col = 1)
```



The "usual" 95% profile confidence interval would thus include zero, so the coefficient is maybe not significant after all. I coded a little function that you can use to find the value at which a profile log-likelihood, here shifted so that the maximum is at zero, intersects with the value of half the χ^2 quantile.

```
# mle <- logLik(huron_arma21)[1]
profile_confint <- function(mle, xvals, profile, lev = 0.95) {
    K <- length(profile)
    ind_max <- which.max(profile)  #value corresponding to MLE
    val_conf <- mle - qchisq(lev, 1)/2  #Cutoff line
```

```
    # Find the closest value by linear interpolation, swapping x and y
    upper <- profile[(ind_max + 1):K]
    lower <- profile[1:ind_max]
    up <- suppressWarnings(c(max(which(upper >= val_conf)), min(which(upper <
        val_conf))) + ind_max)
    low <- suppressWarnings(c(min(which(lower >= val_conf)), max(which(lower <
        val_conf))))
    # Linear interpolation step with two closest values
    if (!any(is.infinite(low))) {
        a <- approx(y = ma1[low], x = profile_loglik[low], xout = val_conf[1])$y
    } else {
        warning("No value of the profile likelihood is below the threshold
           for the lower confidence interval")
        a <- NA
    }
    if (!any(is.infinite(up))) {
        b <- approx(y = ma1[up], x = profile_loglik[up], xout = val_conf[1])$y
    } else {
        warning("No value of the profile likelihood is below the threshold
           for the upper confidence interval")
        b <- NA
    }
    return(c(a, b))
}
# Profile confidence interval
profile_confint(logLik(huron_arma21)[1], xvals = ma1, profile = profile_loglik,
    lev = 0.95)
```

```
[1] -0.5260204         NA
```

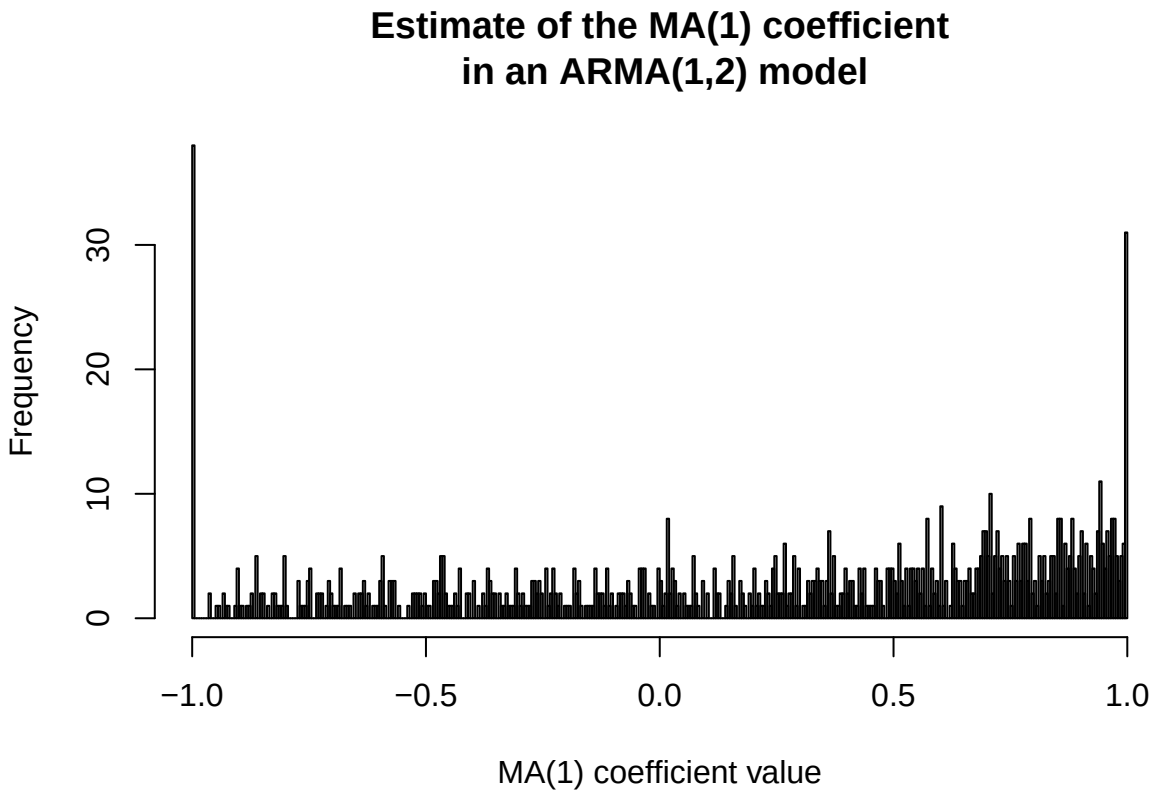
Here, the "usual" 95% confidence interval includes (because of the constraint) all values in (-0.53, 1).

- What do you conclude about the Fisher information confidence interval proposed by `arima`?
- When do you think the Fisher information confidence interval may be reliable?
- Is this profile likelihood plot, and its statistical interpretation, reliable? How do you support your opinion on this?

Be suspicious of the output and think before acting.

There is in fact a non-zero probability of obtaining a coefficient of ± 1 for an MA(1). This can be easily seen through a simulation. We see a picture that roughly matches that of the profile likelihood (noting what seems to be convergence to a local maximum at $\theta = -1$ in many cases).

```
set.seed(54321)
hist(MAcoefdist <- replicate(n=1000, #perform 1000 replicates
      coef(Arima(order=c(2,0,1),  #extract ma[1] coef from ARMA(2,1)
arima.sim(n=155, model=list(ar=coef(huron_ar1)["ar1"])),))["ma1"]), #data is 155 obs from AR(1) model
breaks=300, main="Estimate of the MA(1) coefficient\n in an ARMA(1,2) model", xlab="MA(1) coefficient va
```




```
#Proportion of values that lie on the boundary.
mean(I(abs(MAcoefdist)>0.999))
```

```
[1] 0.069
```

We wish to restrict to causal and invertible processes. The latter implies that the AR(∞) representation of the process exists; the latter is used for prediction. However, while unit roots happen rarely, you only get one sample and it may happen in yours. This makes life more complicated because the asymptotics are non-regular for the problem. It is thus best to rely on procedures that only require estimation of the model under the null hypothesis, such as a score test.

2.3.1 Exercise 3: Lake Erie height

1. Perform an additive decomposition of the form

   $$Y_t = m_t + s_t + Z_t$$

   where m_t is the trend and s_t is a seasonal component of the Lake Erie height series, found at http://sma.epfl.ch/~lbelzile/math342/LakeErie.csv[3]. Characterize the stochastic component Z_t using an ARMA model.
2. Obtain a table of AIC and BIC values for ARMA(p,q) model for order 7 and less. Anything worth of notice?
3. Justifying your answer, select an adequate ARMA model. Are the residuals from this model white noise?
4. Look at the fitted model and check for invertibility and causality of your solution.
5. Plot the (partial) correlogram of Z_t and superpose the theoretical coefficients implied by your model.
6. Plot the correlogram of the residuals of your ARMA model. Do they appear white noise?
7. Perform a Ljung-Box test on the residuals. What can you conclude?

[3] %5Bhttp://sma.epfl.ch/~lbelzile/math342/LakeErie.csv%5D

2.4 Solutions to Exercises

2.4.1 Exercise 1: UBS stock returns

1. Create a function that fits an AR(1)-ARCH(1) model by modifying the code provided above and apply it to `y`.
2. Obtain the maximum likelihood estimates using `nlm` or `optim` as well as the standard errors
3. Make a residual plot. Comment on the fit using standard diagnostic plots (Q-Q plot, ((P)ACF, cumulative periodogram).
4. Fit an AR(2) model using a conditional likelihood for the mean and obtain the standard errors of your estimated coefficients.
5. Perform a likelihood ratio test to test whether the AR(2) coefficient is significative.

```
library(xts)
UBSCreditSuisse <- read.csv("http://sma.epfl.ch/~lbelzile/math342/UBSCSG.csv",
    stringsAsFactors = FALSE)
UBS <- ts(UBSCreditSuisse$UBS_LAST, start = c(2000, 1), frequency = 365.25)
UBS <- ts(UBS[!is.na(UBS)], start = c(2000, 1), frequency = 251.625)
UBS.ret <- 100 * diff(log(UBS))
y <- window(UBS.ret, end = c(2003, 100))
y <- ts(y[-1])
# analysis using AR(1) model for means and ARCH(1) model for variances
nll_AR1_ARCH1 <- function(th, y) {
    n <- length(y)
    condit.mean <- th[1] + th[4] * (y[-n] - th[1])
    condit.var <- th[2] * (1 + th[3] * (y[-n] - th[1])^2)
    -sum(dnorm(y[-1], mean = condit.mean, sd = sqrt(condit.var), log = TRUE))
}
init3 <- c(0, 1, 0.5, 0)
fit3 <- nlm(f = nll_AR1_ARCH1, p = init3, iterlim = 500, hessian = TRUE, y = y)
fit3$minimum
```

```
[1] 1857.097
```

```
fit3$estimate
```

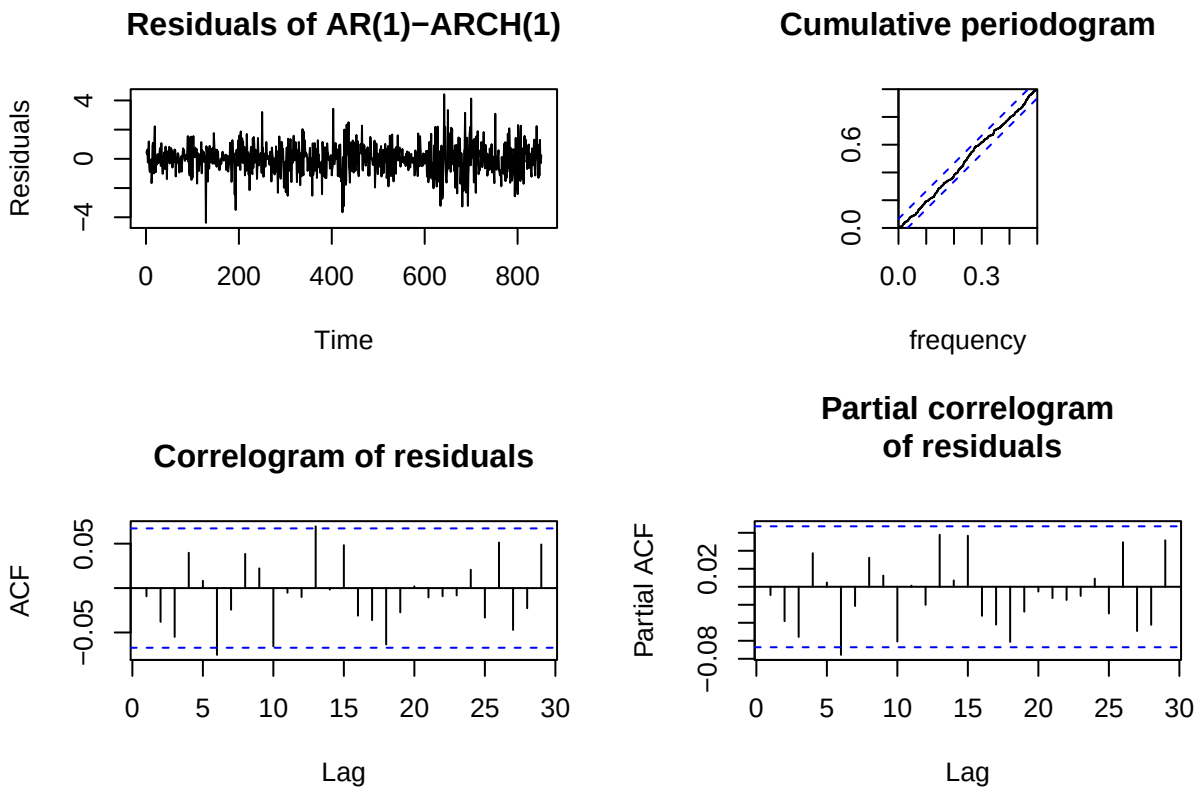
```
[1] 0.03565563 3.30958227 0.11649056 0.08343614
```

```
sqrt(diag(solve(fit3$hessian)))
```

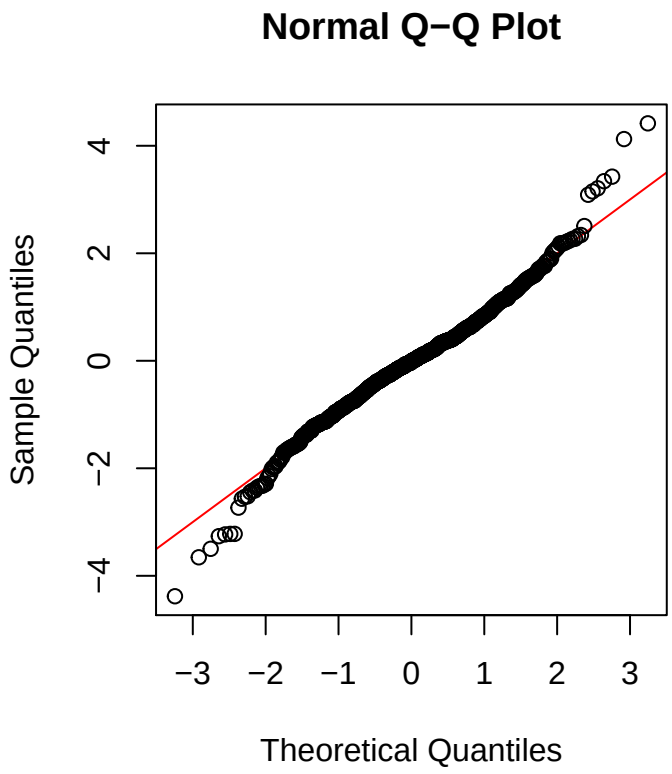
```
[1] 0.07233704 0.24657510 0.02891288 0.04476770
```

```
make_resid_AR1_ARCH1 <- function(y, fit) {
    th <- fit$estimate
    n <- length(y)
    condit.mean <- th[1] + th[4] * (y[-n] - th[1])
    condit.var <- th[2] * (1 + th[3] * (y[-n] - th[1])^2)
    res <- (y[-1] - condit.mean)/sqrt(condit.var)
    ts(res)
}

res3 <- make_resid_AR1_ARCH1(y, fit3)
par(mfrow = c(2, 2))
```

```
plot(res3, ylab = "Residuals", main = "Residuals of AR(1)-ARCH(1)")
cpgram(res3, main = "Cumulative periodogram")
TSA::acf(res3, main = "Correlogram of residuals")
pacf(res3, main = "Partial correlogram\n of residuals")
```




```
par(pty = "s", mfrow = c(1, 1))
qqnorm(res3, panel.first = {
    abline(0, 1, col = "red")
})
```




```
# MLE for the AR(p) using OLS formulation
ARpmle <- function(y, p) {
    dynlm::dynlm(y ~ L(y, 1:p))
}
# If you do it manually with lag, watch out to use lag(y, -1) and NOT lag(y,
# 1) P-value for the likelihood ratio test
llr <- 2 * (logLik(dynlm::dynlm(y ~ L(y, 1:2)))[1] - logLik(dynlm::dynlm(y ~
    L(y, 1), start = 3))[1])
# Fail to reject the null that the simpler model is adequate.
1 - pchisq(llr, df = 1)
```

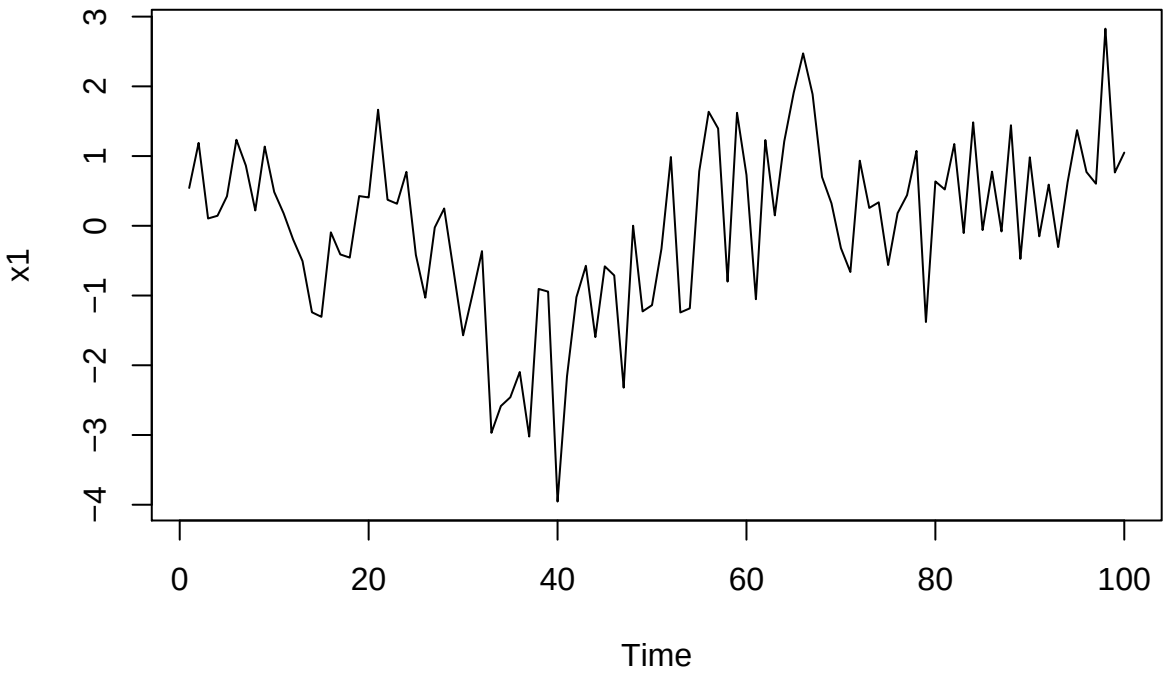
```
[1] 0.1230465
```

2.4.2 Exercise 2: Simulated series

Warning The following output is unformatted and rather sketchy, but should illustrate development to find an adequate model

1. The `Simulated_ARMA` dataset contains 8 time series (these examples are from Charlotte Wickham's website). Fit ARMA models to those and select a model, justifying your choice. Be skeptical of the optimization routine and analyze the models carefully.

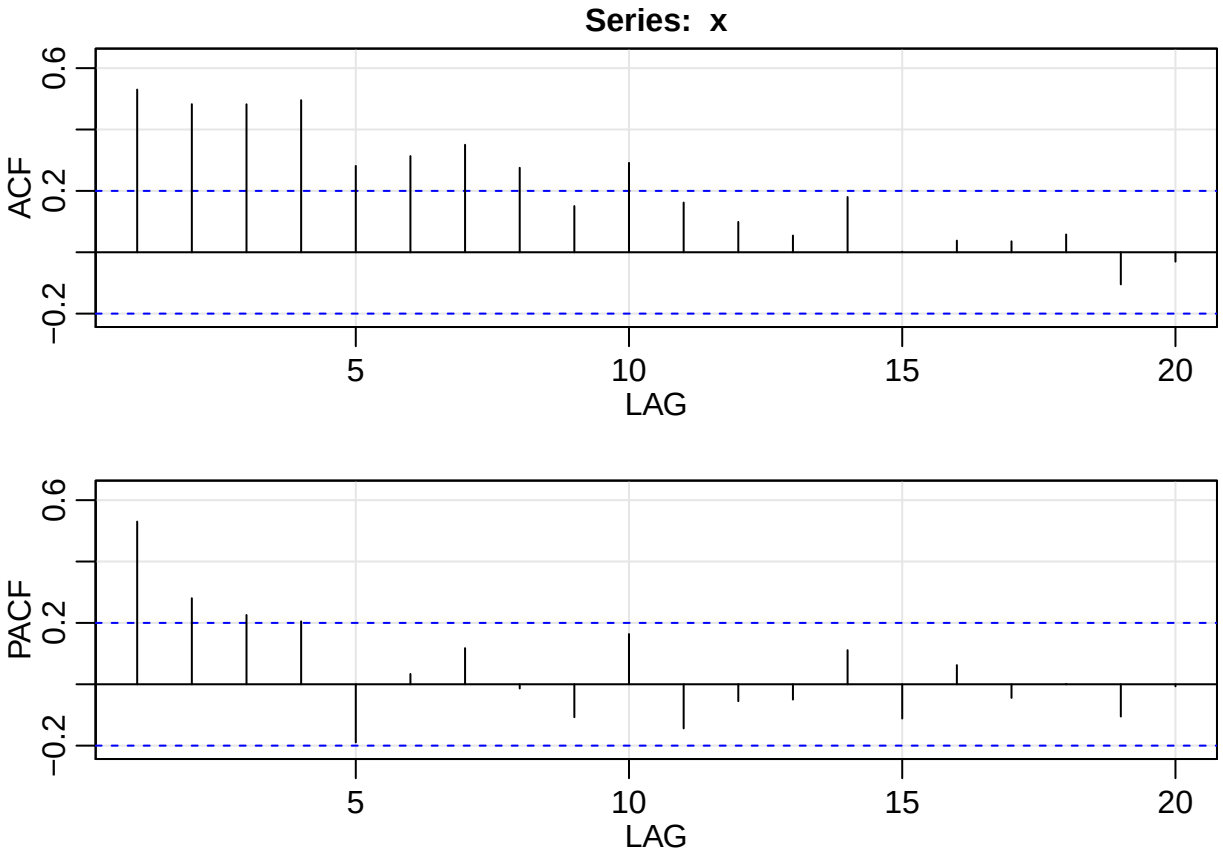
```
library(forecast)
library(astsa)
load(url("http://sma.epfl.ch/~lbelzile/math342/Simulated_ARMA.RData"))
plot(x1)
```



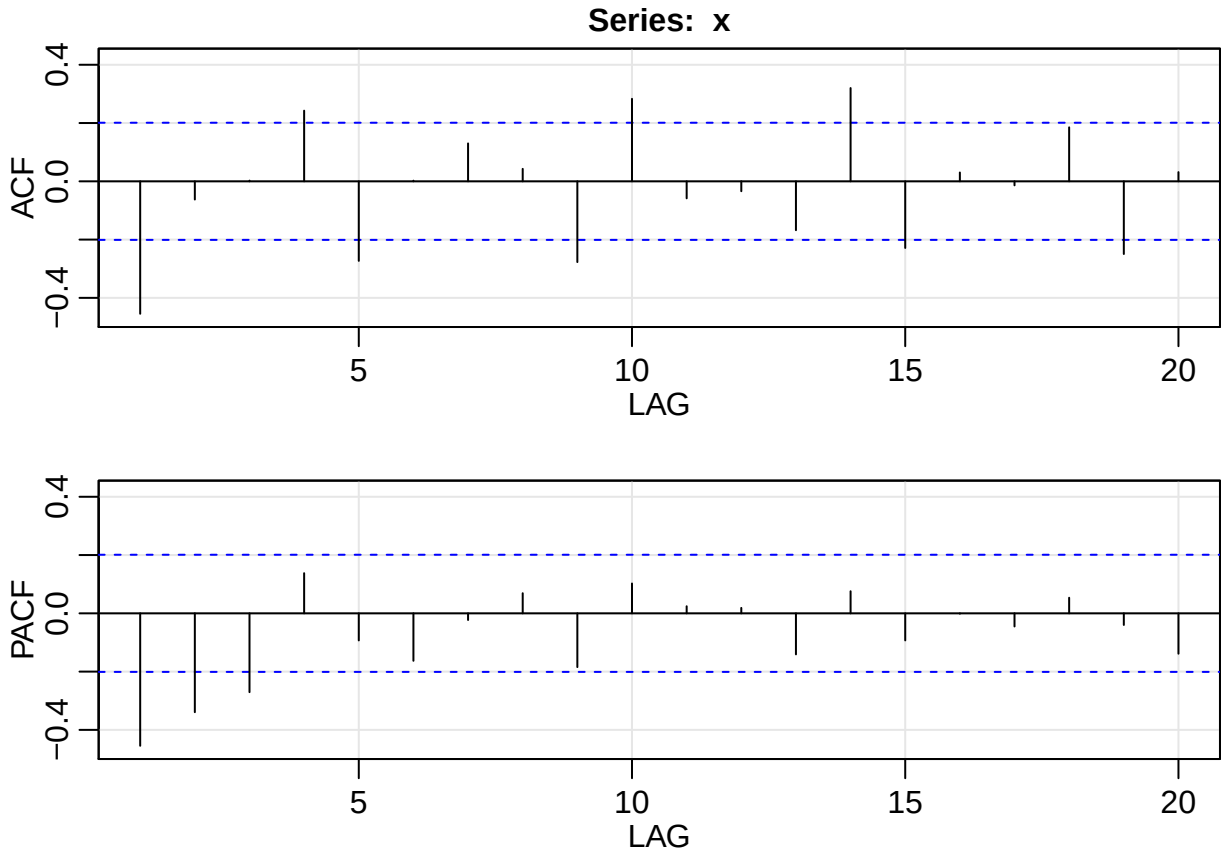
The first series, x_1 has a visible trend and may not be stationary, even if the series looks like Gaussian. After first differencing (setting the component $d = 1$ corresponds to first differencing), one could think that an AR(2) model is appropriate by looking at the ACF/PACF. Even a simple MA(1) will do the job here, and it is more parsimonious.

For the second, a simple MA(1) model also is sufficient for removing the linear autocorrelation.

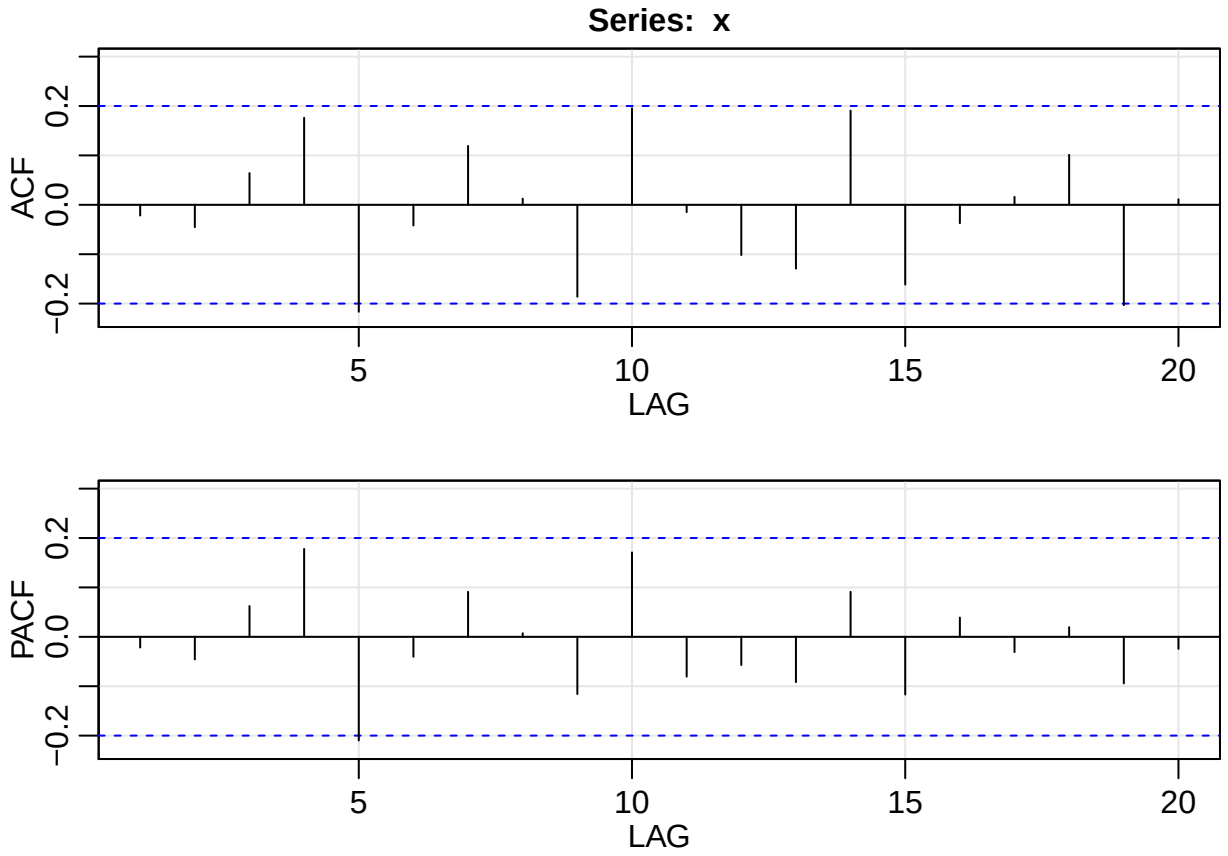
```
myacf2 <- function(x) {
    invisible(capture.output(astsa::acf2(x)))
}
myacf2(x1)
```



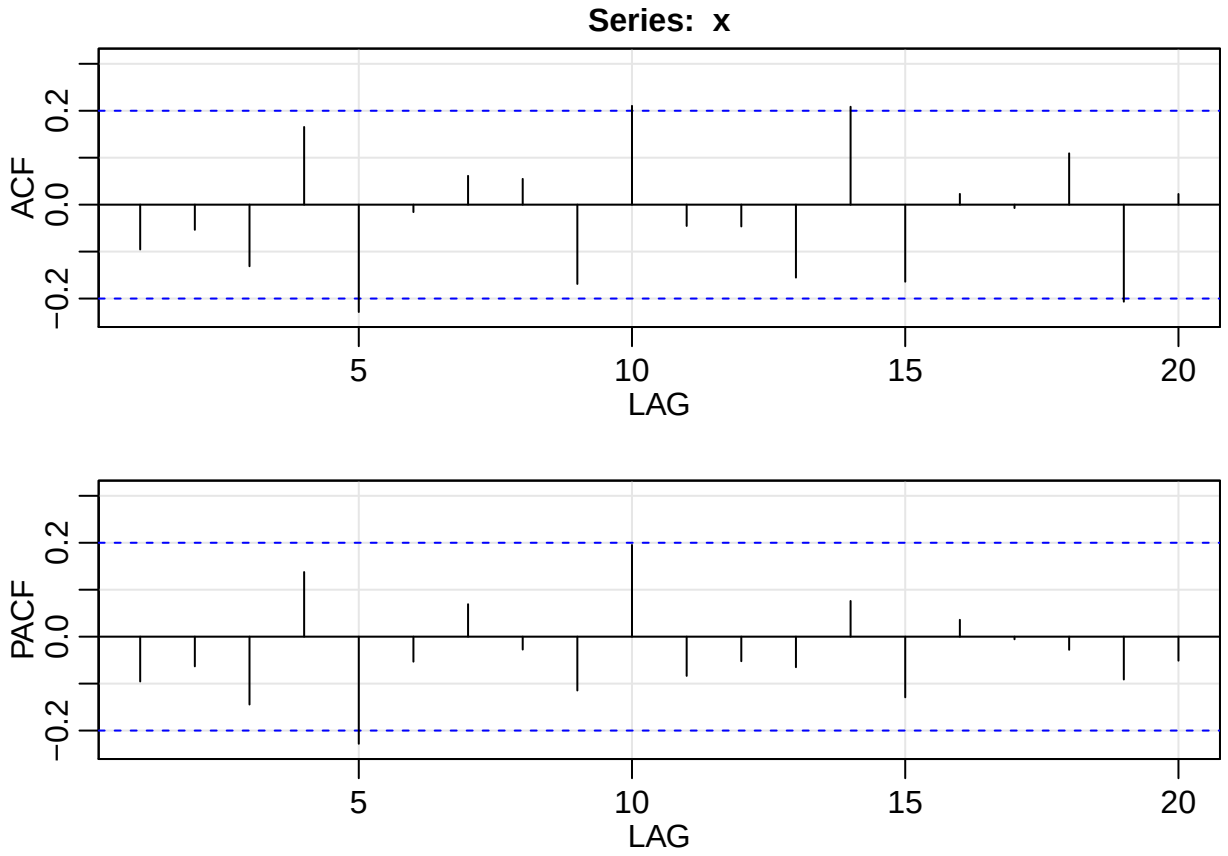

```
myacf2(diff(x1))
```




```
arima_1_011 <- Arima(x1, order = c(0, 1, 1))
arima_1_210 <- Arima(x1, order = c(2, 1, 0))

myacf2(resid(arima_1_011))
```




```
myacf2(resid(arima_1_210))
```




```
BIC(arima_1_011)
```

```
[1] 283.7835
```

```
BIC(arima_1_210)
```

```
[1] 291.6321
```

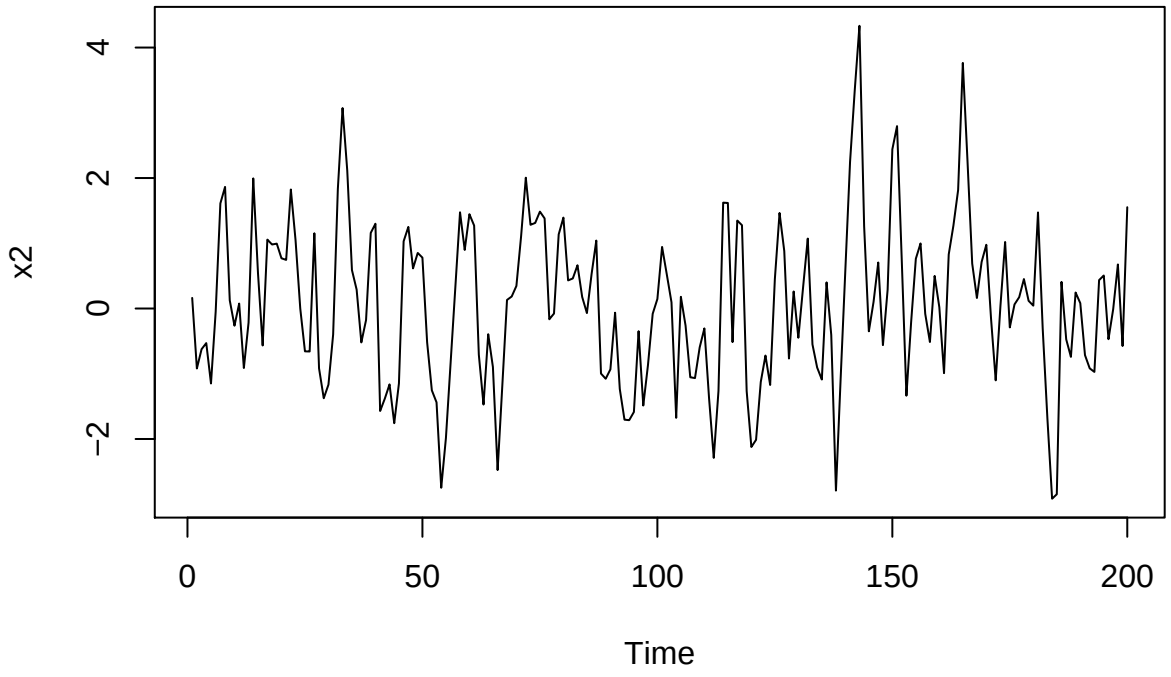
```
AIC(arima_1_011)
```

```
[1] 278.5933
```

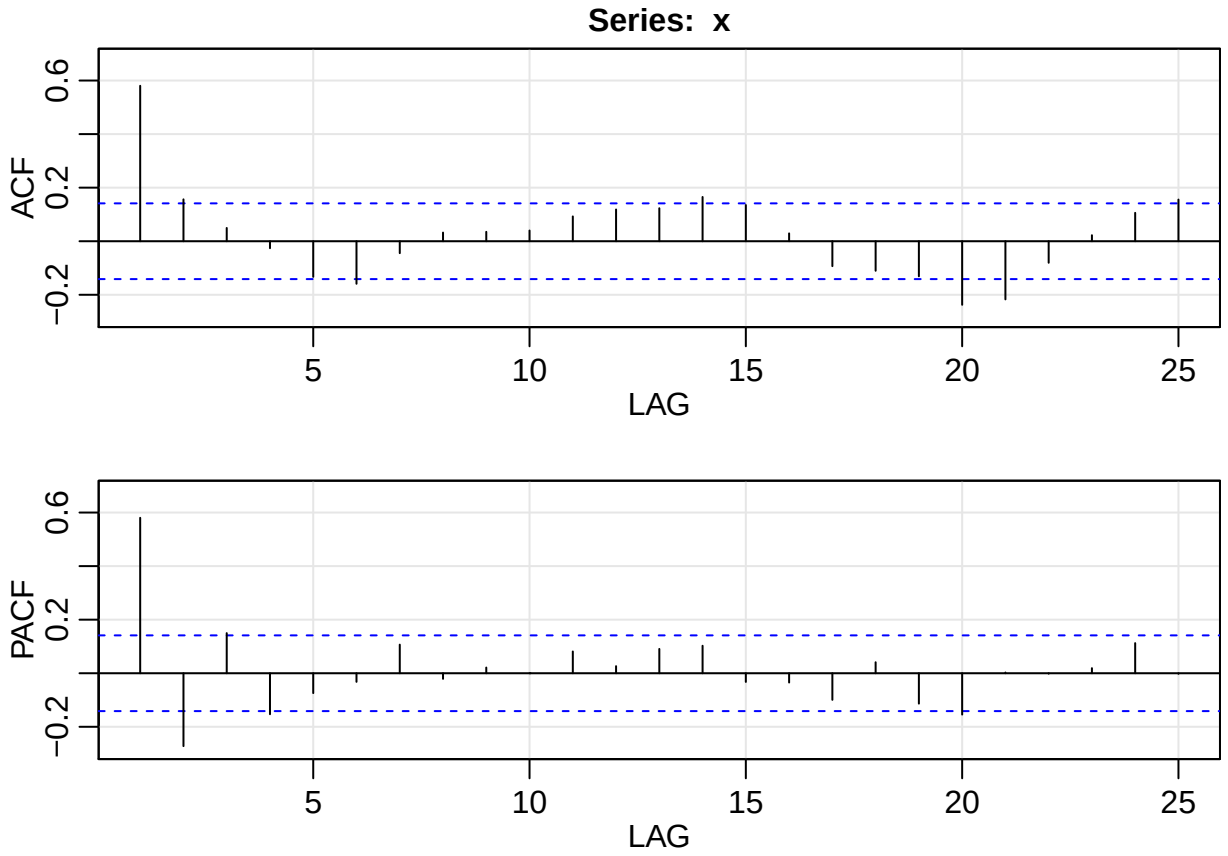
```
AIC(arima_1_210)
```

```
[1] 283.8468
```

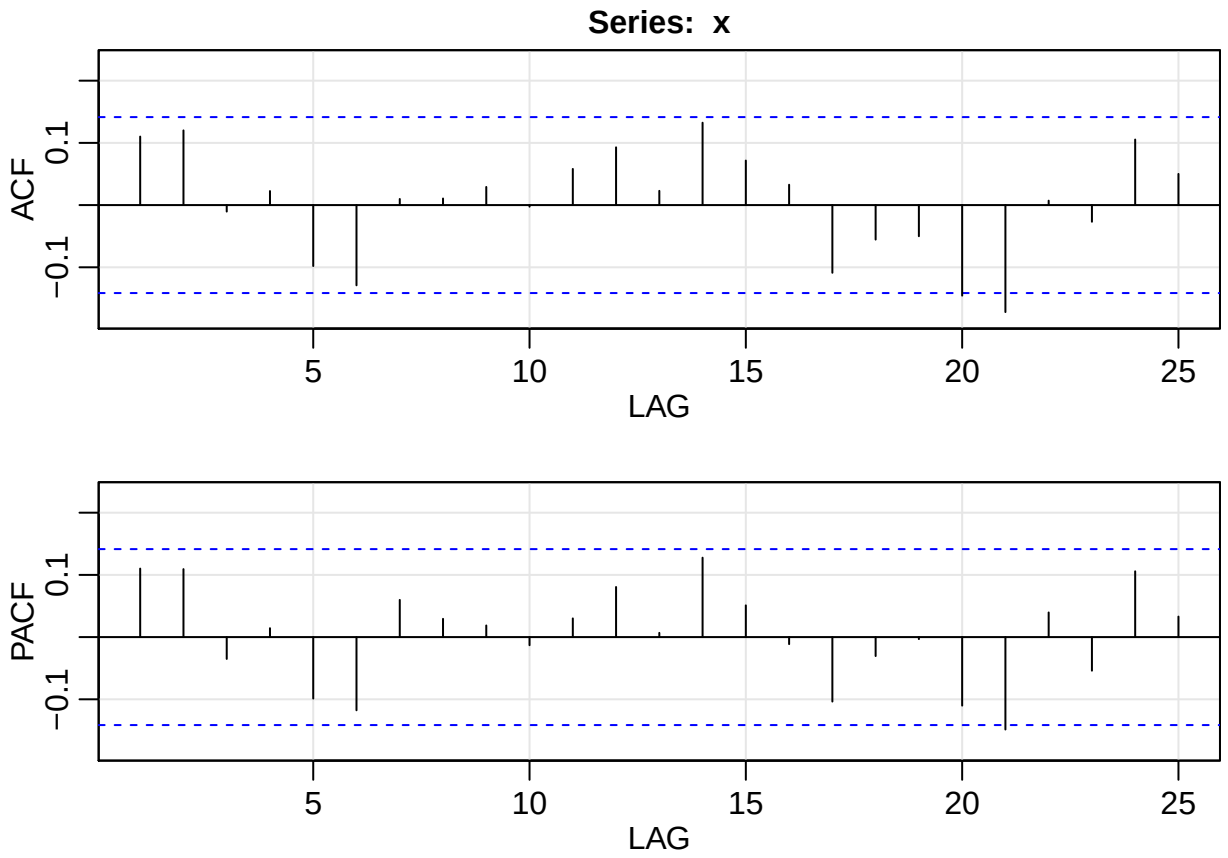
```
plot(x2)
```




```
myacf2(x2)
```



```
arima_2_001 <- Arima(x2, order = c(0, 0, 1))
myacf2(resid(arima_2_001))
```



The third series x_3 is difficult. Indeed, there is a trend in the data (or else non-stationarity that requires differencing). If only autoregressive components are included, they fail to capture the structure adequatly. If only one MA component is included, the process is often estimated to be non-invertible. This excludes the ARMA(1,1), moany differenced series whose MA polynomial has order 1 such as the ARMA(0,1) and ARMA(1,1). An ARMA(1,2) or an ARMA(2,1) with a trend fits the data appropriately, or else an ARMA(2,1) for the differenced series.

x_4 likely corresponds to an AR(1) process and is fairly obvious from the plot of the PACF and the oscillation in the correlogram. The series in x_5 seem to have exhibit some heteroscedasticity. If we take first difference, we get a partial correlogram that is reminiscent of AR(1). As in many cases, a MA(1) with a negative component fits equally well.

As mentionned before, the following is rather rough output and is not formatted (graphs have no labels, etc.) It does however shows the sort of workflow: start from simple and build up more complicated models. You can work out for yourself the most likely data generating processes of x_6, x_7 and x_8.

```
# Here, evidence trend or nonstationarity
plot(x3)
myacf2(x3)
# with trend
arima_3_101trend <- Arima(x3, order = c(1, 0, 1))
arima_3_200trend <- Arima(x3, order = c(2, 0, 0))
arima_3_300trend <- Arima(x3, order = c(3, 0, 0))
arima_3_201trend <- Arima(x3, order = c(2, 0, 1))
```

```
arima_3_102trend <- Arima(x3, order = c(1, 0, 2))
# after differencing
arima_3_011 <- Arima(x3, order = c(0, 1, 1))
arima_3_111 <- Arima(x3, order = c(1, 1, 1))
arima_3_211 <- Arima(x3, order = c(2, 1, 1))
# ARIMA models are not invertible
plot(arima_3_101trend)
plot(arima_3_011)
plot(arima_3_111)
plot(arima_3_211)  #still on the boundary
# AR2 model still has some structure
myacf2(resid(arima_3_200trend))
myacf2(resid(arima_3_300trend))
# ARMA(2,1) is okay, but not invertible
myacf2(resid(arima_3_201trend))
plot(arima_3_201trend)
# Could also difference ARMA(2,1) is okay if trend, so is first differenced
# ARMA(1,1) or ARIMA(0,1,1)
myacf2(resid(arima_3_011))
```

2.4.3 Exercise 3: Lake Erie height

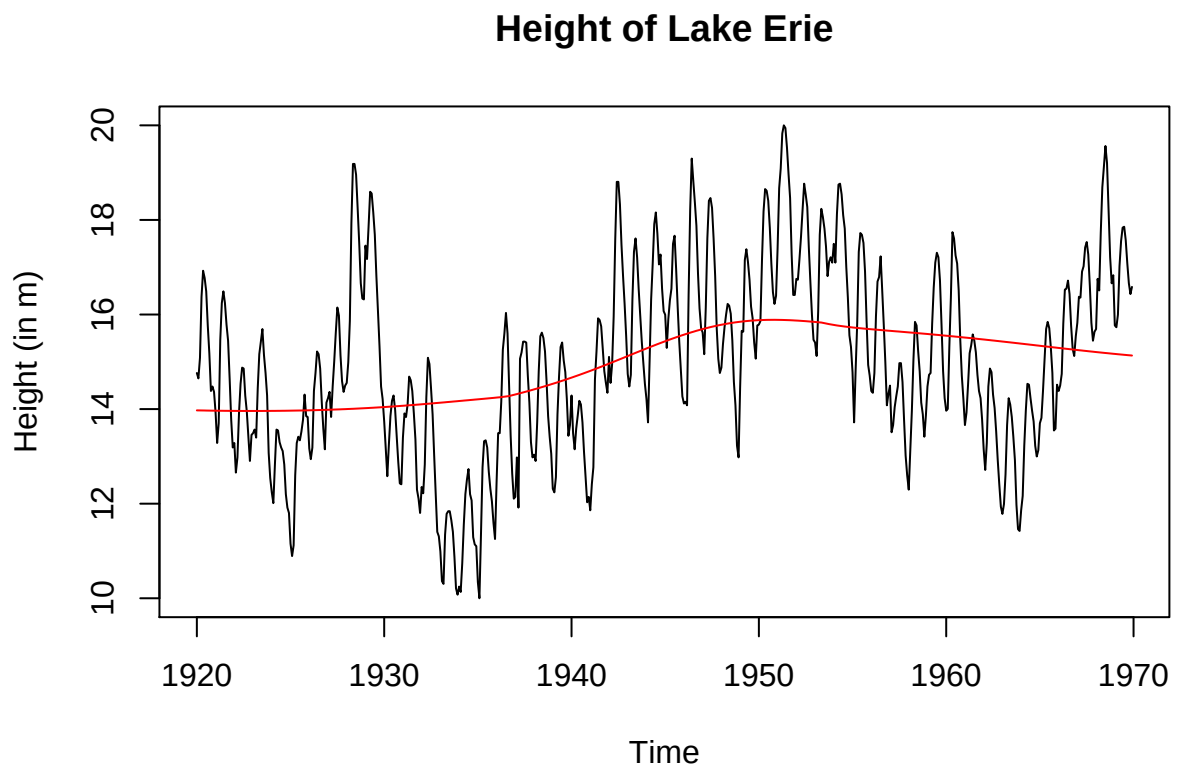
1. Perform an additive decomposition of the form

$$Y_t = m_t + s_t + Z_t$$

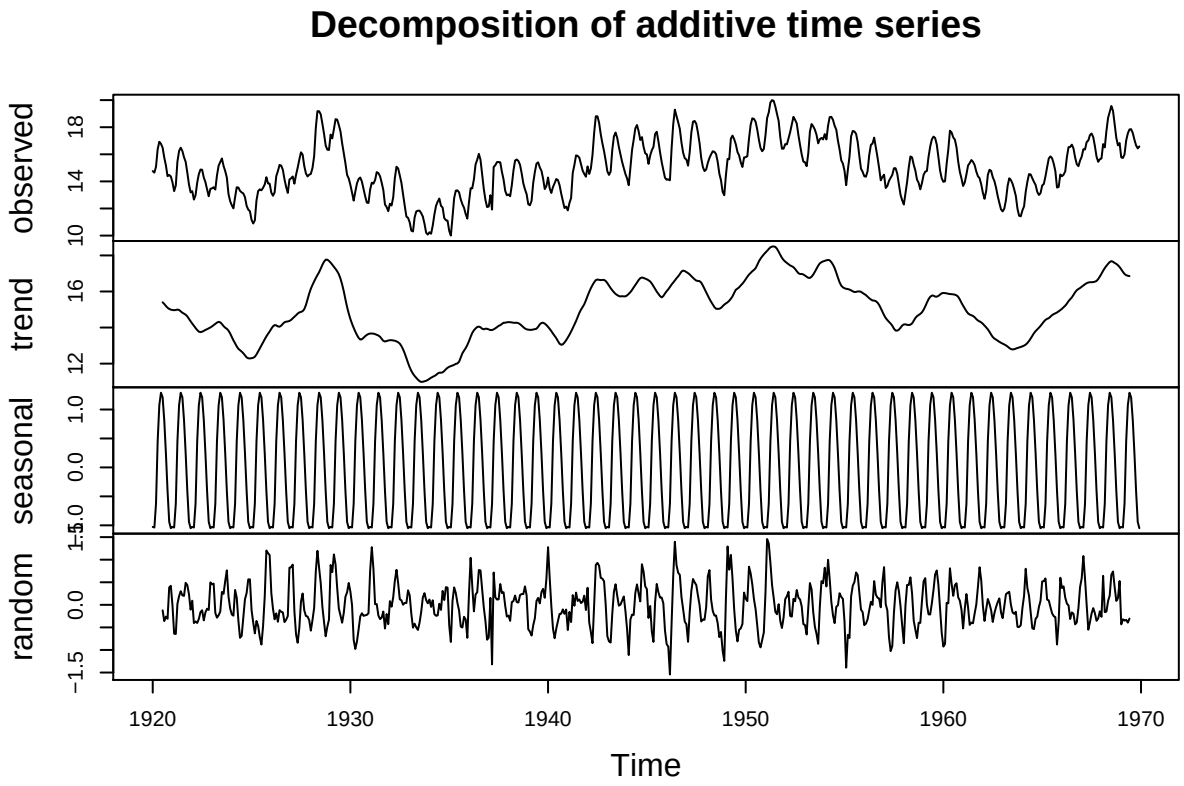
   where m_t is the trend and s_t is a seasonal component of the Lake Erie height series, found at http://sma.epfl.ch/~lbelzile/math342/LakeErie.csv[4]. Characterize the stochastic component Z_t using an ARMA model.
2. Obtain a table of AIC and BIC values for ARMA(p,q) model for order 7 and less. Anything worth of notice?
3. Justifying your answer, select an adequate ARMA model. Are the residuals from this model white noise?
4. Look at the fitted model and check for invertibility and causality of your solution.
5. Plot the (partial) correlogram of Z_t and superpose the theoretical coefficients implied by your model.
6. Plot the correlogram of the residuals of your ARMA model. Do they appear white noise?
7. Perform a Ljung-Box test on the residuals. What can you conclude?

```
library(forecast)
lake <- read.csv("http://sma.epfl.ch/~lbelzile/math342/LakeErie.csv", header = TRUE,
    stringsAsFactors = FALSE, sep = ";")

lake.ts <- ts(lake[, 2], start = c(1920, 1), frequency = 12)
plot(lake.ts, ylab = "Height (in m)", main = "Height of Lake Erie")
lake.lowess <- lowess(lake[, 2])
lines(as.vector(time(lake.ts)), lake.lowess$y, col = 2)
```

[4] %5Bhttp://sma.epfl.ch/~lbelzile/math342/LakeErie.csv%5D




```
detrended <- decompose(lake.ts)
plot(decompose(lake.ts))
```



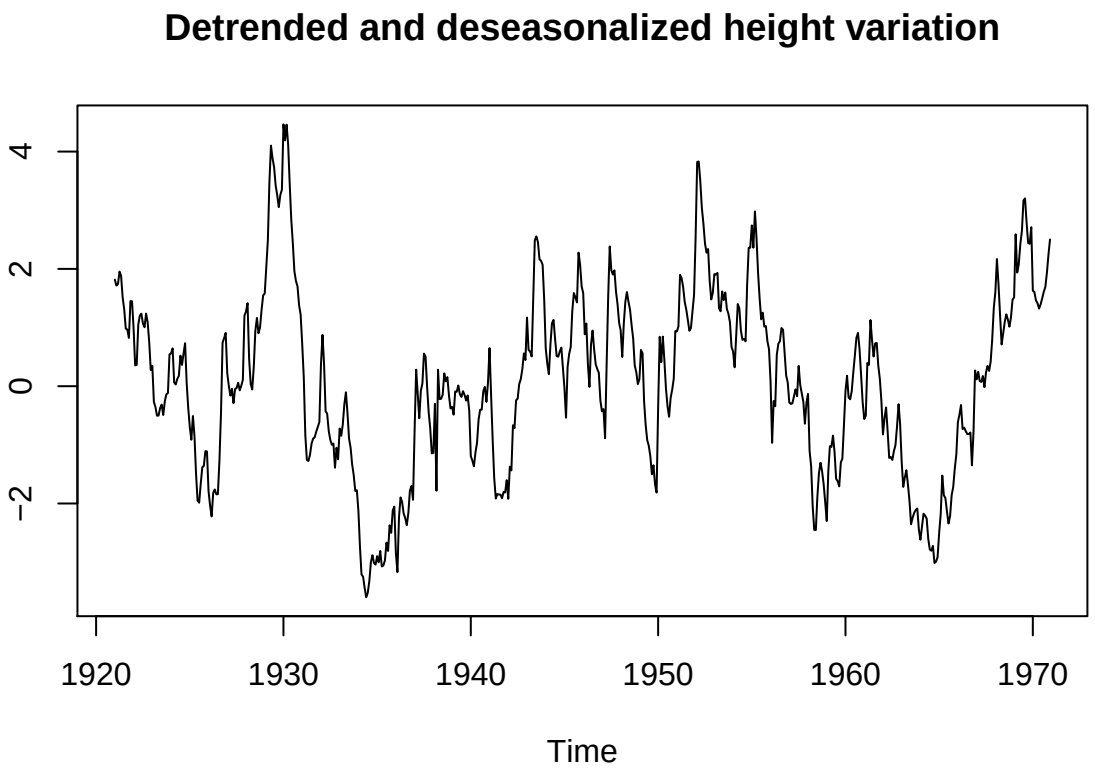
It is clear from the picture that the series exhibit a strong yearly variation, which should be taken into account.

The trend decomposition seems somewhat more arbitrary. After deaseasonalizing, it appears that there may be a structural break around 1950, from which the trend onward for the lake level is decreasing linearly. This could be assesed formally by e.g. a Chow test; instead, to deal with this, we use a non-parametric polynomial fit to remove the (nonlinear) trend and demean the data before attempting to use an ARMA model on the residuals. Thus, we are taking the original series X_t and decompose it into

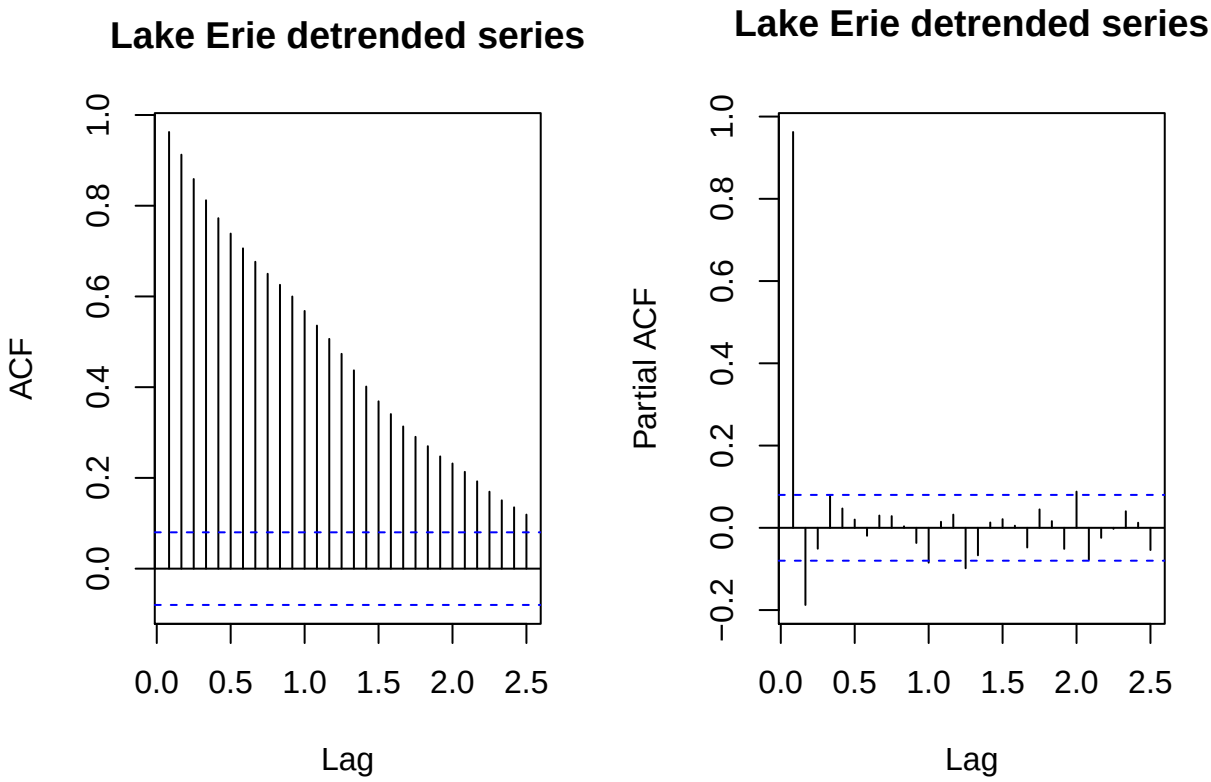
$$X_t = m_t + s_t + Y_t$$

where plausibly $Y_t \sim \text{ARMA}(p, q)$.

```
# plot(stl(lake.ts, s.window='periodic'))
lake.detrended <- ts(lake[, 2] - lake.lowess$y - detrended$seasonal, start = c(1921,
    1), frequency = 12)
plot(lake.detrended, ylab = "")
title("Detrended and deseasonalized height variation")
```




```
par(mfrow = c(1, 2))
TSA::acf(lake.detrended, main = "Lake Erie detrended series", lag.max = 30)
pacf(lake.detrended, main = "Lake Erie detrended series", lag.max = 30)
```

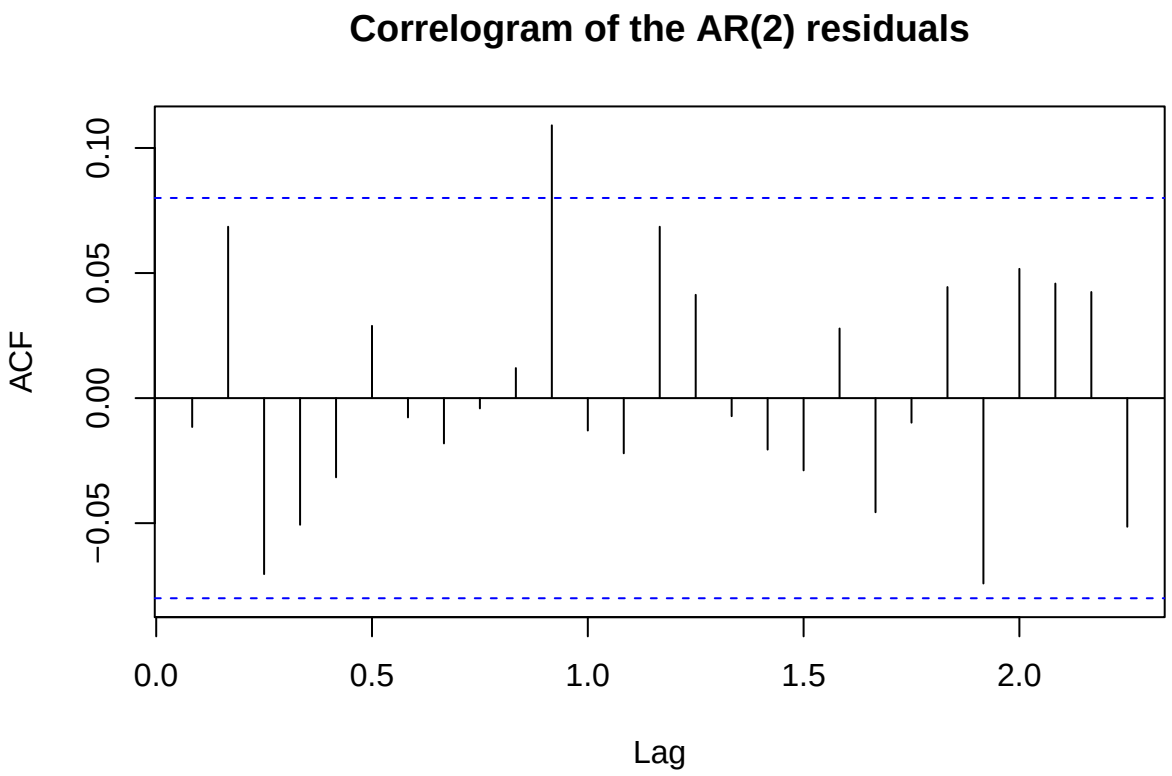


It is apparent from the observation of the ACF that there is remaining structure for the data. We attempt to model this using an ARMA model, using the BIC and the AICc, respectively the Bayesian and bias-corrected information criterion for the data, as guiding mesure of data complexity. The selection yields an AR(2) as privilegied model with the BIC and the AICc and the somewhat more complex ARMA(6,4) from AIC.

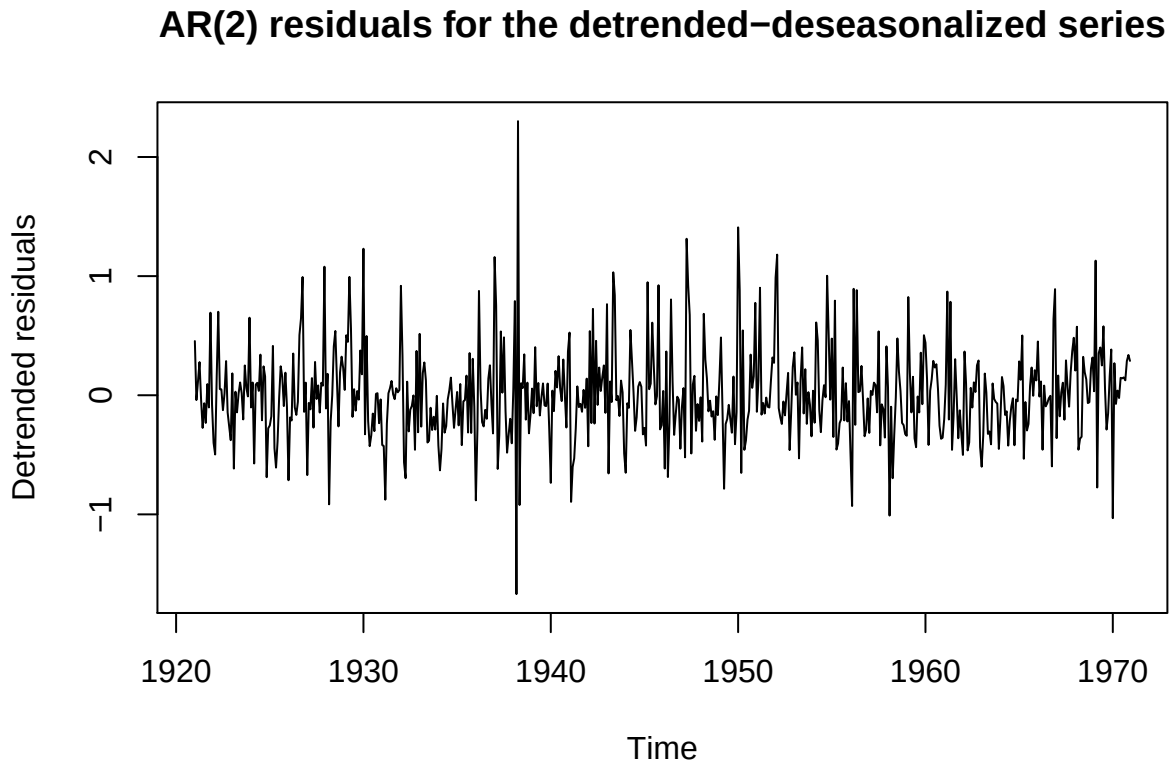
```
AIC.vals <- AICc.vals <- BIC.vals <- matrix(0, 8, 8)
N <- length(lake.detrended)
for (p in 0:7) {
    for (q in 0:7) {
        f <- Arima(lake.detrended, order = c(p, 0, q), include.mean = FALSE,
            method = "ML", optim.control = list(maxit = 1000))
        # Defn of AICc is a correction of (2*(p+q)*(p+q+1))/(N-p-q-2) from AIC
        AIC.vals[p + 1, q + 1] <- f$aic
        AICc.vals[p + 1, q + 1] <- f$aic + (2 * (p + q + 2) * (p + q + 1))/(N -
            p - q - 2)  #or f$aicc
        BIC.vals[p + 1, q + 1] <- BIC(f)  #or f$bic
        # also AIC(arima(..., k=log(length(lake.detrended)))) for the BIC
    }
}
```

The ARMA(6, 4) model has smallest AIC, the ARMA(2, 0) model has smallest BIC and ARMA(6, 4) model has smallest AIC_c The PACF here would have provided rough guidelines indicating that an AR(2) was indeed appropriate. Let us compare the two models.

```
# Model selection with deaseasonalized-detrended series ARMA(6,4) from AIC
# and AICc; AR(2) from BIC Model selected through BIC
ar2 <- Arima(lake.detrended, order = c(2, 0, 0), include.mean = FALSE, optim.control = list(maxit = 100
TSA::acf(residuals(ar2), main = "Correlogram of the AR(2) residuals", ylab = "ACF")
```




```
plot(residuals(ar2), main = "AR(2) residuals for the detrended-deseasonalized series",
    ylab = "Detrended residuals")
```



```
summary(ar2)
```

```
Series: lake.detrended
ARIMA(2,0,0) with zero mean

Coefficients:
         ar1      ar2
      1.1692  -0.2095
s.e.  0.0399   0.0400

sigma^2 estimated as 0.1549:  log likelihood=-292.28
AIC=590.57   AICc=590.61   BIC=603.76

Training set error measures:
                      ME      RMSE       MAE      MPE     MAPE      MASE
Training set 0.003821393 0.3929215 0.2848819 22.00608 91.12439 0.2483707
                    ACF1
Training set -0.01154223
```

```
ar2.acf.fit <- ARMAacf(ar = ar2$coef[1:2], lag.max = 30)
Mod(polyroot(c(1, -ar2$coef[grep("^ar", names(ar2$coef))])))
```

```
[1] 1.054612 4.525466
```

```
# Model selected through AIC and AICC
arma64 <- arima(lake.detrended, order = c(6, 0, 4), include.mean = FALSE, optim.control = list(maxit = 
arma64.par <- arma64$coef
summary(arma64)
```

```
Call:
arima(x = lake.detrended, order = c(6, 0, 4), include.mean = FALSE, optim.control = list(maxit = 1000))

Coefficients:
         ar1      ar2     ar3      ar4     ar5      ar6     ma1     ma2
      0.9479  -1.5285  1.6102  -0.8386  0.7140  -0.0511  0.2146  1.6741
s.e.  0.1506   0.2185  0.1873   0.2321  0.1657   0.0522  0.1430  0.0788
         ma3     ma4
      0.1281  0.7901
s.e.  0.1178  0.1036

sigma^2 estimated as 0.1477:  log likelihood = -279.62,  aic = 581.24

Training set error measures:
                      ME      RMSE       MAE     MPE     MAPE      MASE
Training set 0.003804229 0.3842917 0.2821519 20.0979 97.20178 0.9433899
                    ACF1
Training set -0.0003748186
```

```
arma64.acf.fit <- ARMAacf(ar = arma64.par[1:6], ma = arma64.par[7:10], lag.max = 30)
Mod(polyroot(c(1, -arma64$coef[grep("^ar", names(arma64$coef))])))
```

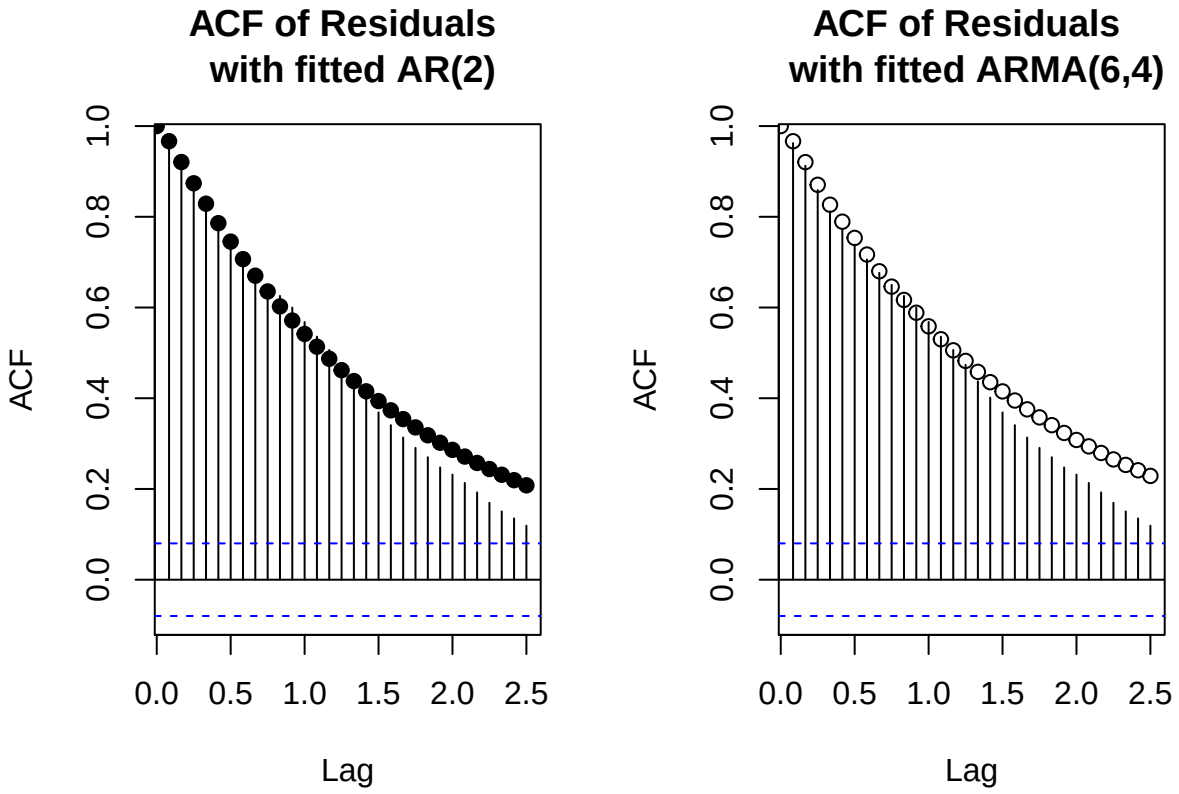
```
[1]  1.050757  1.013319  1.013319  1.186975  1.186975 12.866561
```

```
Mod(polyroot(c(1, arma64$coef[grep("^ma", names(arma64$coef))])))
```

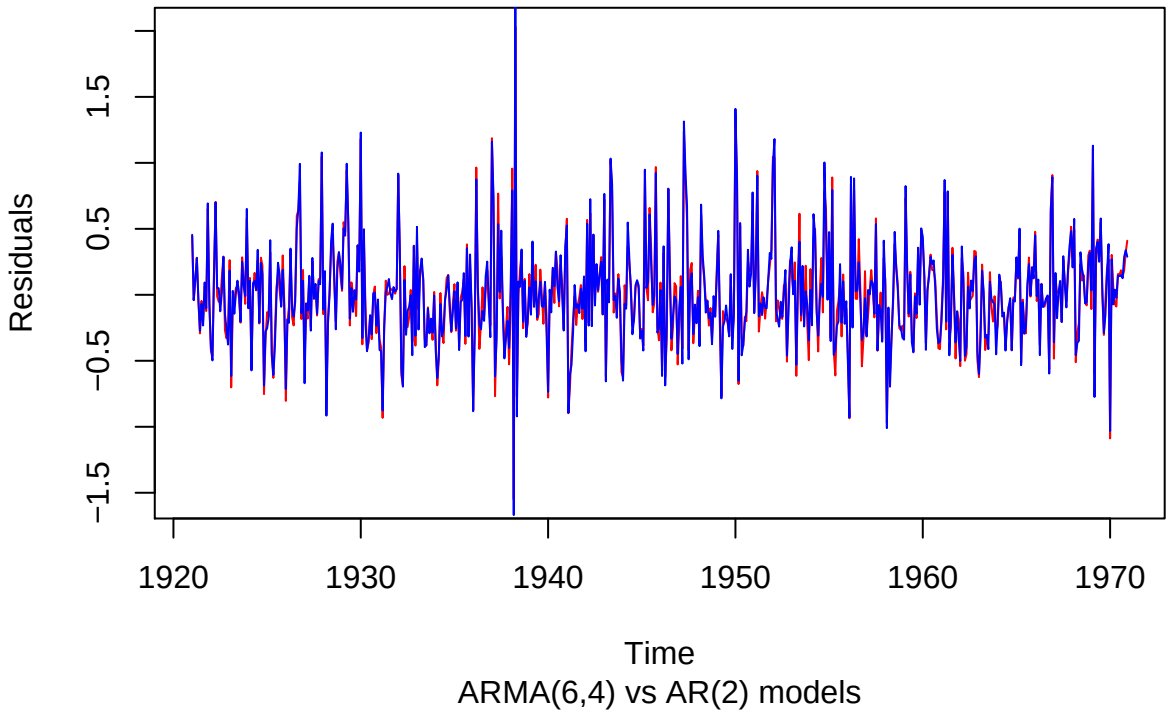
```
[1] 1.115831 1.008236 1.008236 1.115831
```

Here, the estimate from `arima` for the ARMA(6,4) did not adjust the AR coefficients. The process is barely causal, as some roots are close to the unit circle – it would be advisable to differentiate the series. It is also barely invertible, as one of the roots is transformed to 1.

```
par(mfrow = c(1, 2))
TSA::acf(lake.detrended, lag.max = 30, main = "ACF of Residuals \n with fitted AR(2)")
points(seq(0, 2.5, length.out = 31), ar2.acf.fit, pch = 19)
TSA::acf(lake.detrended, lag.max = 30, main = "ACF of Residuals \n with fitted ARMA(6,4)")
points(seq(0, 2.5, length.out = 31), arma64.acf.fit)
```




```
# Plot of the residuals of the series
par(mfrow = c(1, 1))
plot(ts(residuals(arma64), start = c(1921, 1), frequency = 12), ylab = "Residuals",
    sub = "ARMA(6,4) vs AR(2) models", col = 2)
lines(ts(residuals(ar2), start = c(1921, 1), frequency = 12), ylab = "Residuals",
    col = 4)
```



The ARMA(6,4) regression model has lower variance, but it is rather clear that it won't be the most parsimonious (and the forecasting performances could have been worst, but here there is little difference).

```
# Ljunx-Box test for independence of residuals, proposed 22 lags frame
# suggested by B&D Adjusting for the number of parameters in the ARMA model
Box.test(residuals(ar2), lag = 22, fitdf = 2)
```

```
    Box-Pierce test

data:  residuals(ar2)
X-squared = 23.942, df = 20, p-value = 0.2449
```

```
Box.test(residuals(arma64), lag = 22, fitdf = 10)
```

```
    Box-Pierce test

data:  residuals(arma64)
X-squared = 9.3147, df = 12, p-value = 0.6758
```

The correlogram plot show convincing evidence that the autocorrelation structure has been mostly captured by the model, although some lag at yearly intervals appears to be still present. The residuals from Y_t, although may not be constant in variance, appear plausibly white-noise. We can test for assumption of white noise; using the Ljung-Box test, we fail in both cases to reject the null of linear independence of the residuals. It is clear that the data is not very stationary, and that a seasonal model would do better here. This we shall see in the next weeks, with SARIMA models.

Note: An ARMA(4, 1) would fit well the data if you started with the residuals from `decompose` or `stl`.

Chapter 3

Seasonal ARIMA and GARCH models

This tutorial addresses the following: - estimation and forecasting for SARIMA models. - uncertainty quantification using the bootstrap for time series. This material is optional. - estimation of GARCH and forecast from the latter using rolling-windows.

3.1 SARIMA models: estimation and forecasting

We have covered the estimation of ARIMA model in the last tutorial. Extension to SARIMA models is immediate: simply use - argument `seasonal` (a vector of length 3) for `forecast::Arima` - arguments `P`, `D`, `Q` and the period `S` in the function `astsa::sarima` - a list `seasonal` with components `order` and `period` (the latter is facultative if supplying `ts` objects) for `stats::arima`.

Estimation is done using a state-space formulation of the model; an alternative would be to fit the parameters using an artificial regression[1].

We have yet to address the matter of prediction. The latter proceeds by one-step ahead forecasting and emploies again the state-space representation. Note however that

> The standard errors of prediction exclude the uncertainty in the estimation of the ARMA model and the regression coefficients. According to Harvey (1993, pp. 58–9) the effect is small.

The `predict.Arima` method gives forecast for h-lags ahead for `arima` objects. For objects of class `Arima`, the `forecast` function from the eponym package handles the additional features from `forecast::Arima`, namely Box–Cox transformation viz `lambda` and bootstrap prediction intervals, as a logical supplied to the argument `bootstrap` (more later on this). If you use `astsa::sarima`, the forecast function `astsa::sarima.for` allows you to do estimation and prediction all in one go.

##An aside on models with regressors (optional)

Rather than performing your inference in multiple steps (detrending, then modelling the residuals), it is sometimes better to do all at once to correctly characterize the uncertainty arising from estimation. You can do so in a regression context by using a generalized least squares specification.

That is, we can consider fitting a model of the form

$$\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where $\mathbf{X}$ is assumed to be non-stochastic, $\mathsf{E}(\boldsymbol{\varepsilon}) = 0$ and $\mathsf{Var}(\boldsymbol{\varepsilon}) = \boldsymbol{\Omega}$.

[1] http://russell-davidson.arts.mcgill.ca/e761/mlzero3.pdf

The GLS estimator that solves the minimization problem for correlated errors is

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \boldsymbol{\Omega}^{-1}\mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{\Omega}^{-1}\boldsymbol{y}$$

and one can easily show it is best linear unbiased predictor (BLUE). The function `gls` in **R** has an argument `correlation`. One can specify `corARMA(p, q)` to provide the structure of the errors, but the fitting procedure is time-consuming and may fail to converge.

The `arima` (and variants thereof) function allows one to specify a model whose errors follow an ARMA model, viz.

$$y_t = \mathbf{X}_t\boldsymbol{\beta} + \eta_t, \qquad \eta_t = \sum_{j=1}^{p} \phi_j \eta_{t-j} + \sum_{k=1}^{q} \theta_k \varepsilon_{t-k} + \varepsilon_t.$$

This is a somewhat modern variant of the Cochran–Orcutt procedure.

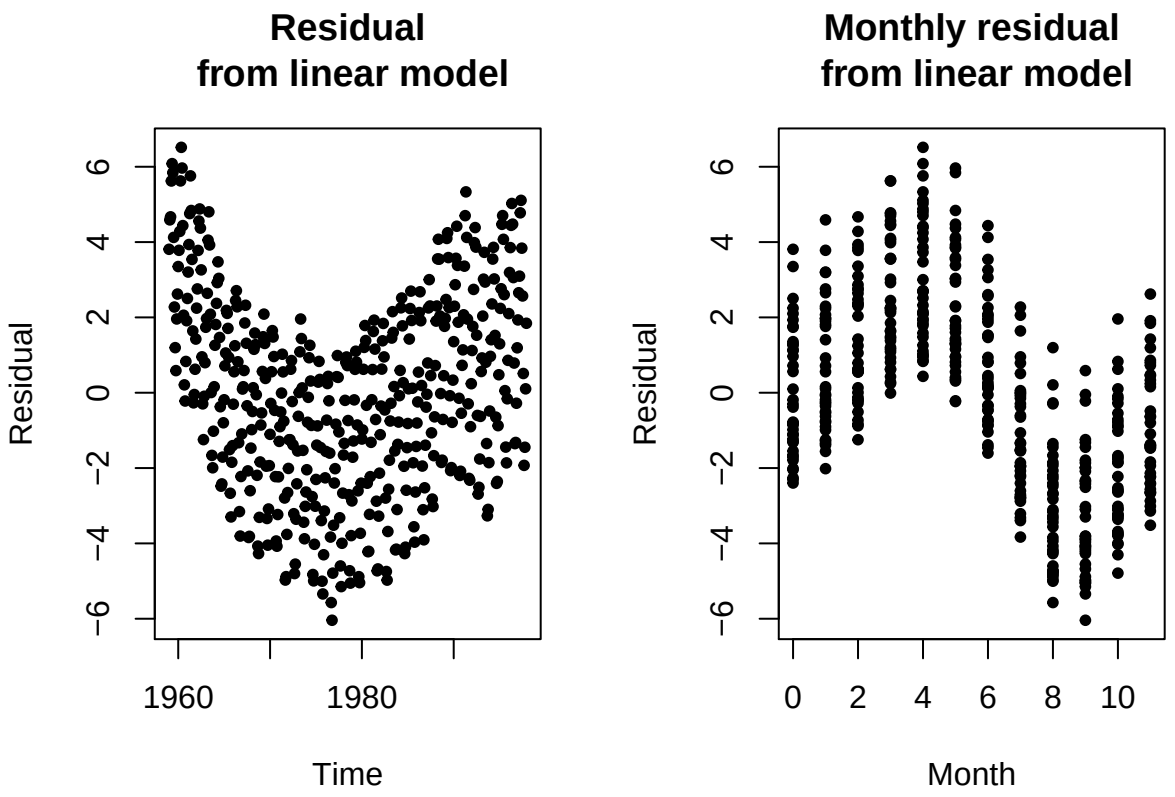
A matrix of external regressors can be supplied to any of `arima` functions via the argument `xreg`. There are subtleties associated with fitting time series models with regressors. See notably The ARIMAX model muddle by Rob Hyndman[2].

3.1.1 Mauna Loa CO_2 dataset

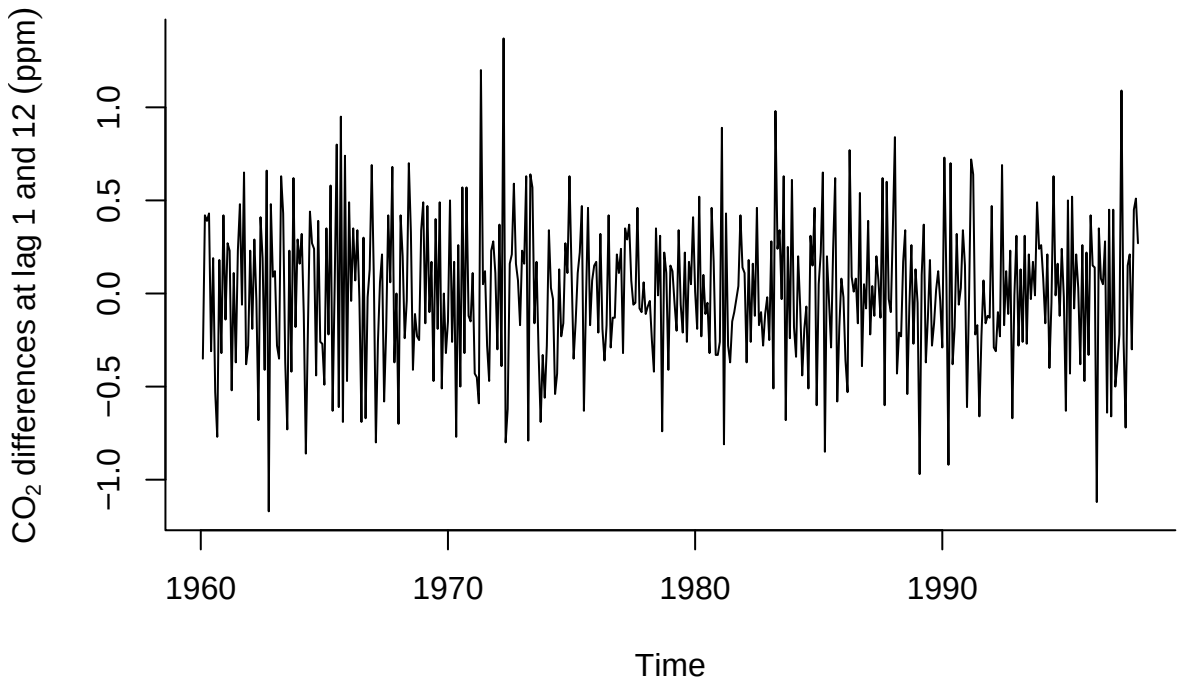
We revisit the Mauna Loa CO_2 example. While one can see very clear trend and seasonality. One could in principle add a linear trend to the model. Detrending is difficult because there is clear evidence that the growth is exponential and not linear. One could nevertheless fit a model to the deterministic part and then model residuals. This would likely require removing the components $(1-B)(1-B^{12})$ from the model, but may create artifacts that will be due to model misspecification. The SARIMA prediction seemingly works because of the combination of the seasonal unit root and the unit root (note however how the observed pattern from the last cycle gets reproduced).

```
data("co2", package = "datasets")
# Look at the deterministic structure of the model, ignoring the time
# component
n <- length(co2)
month <- as.vector(round((time(co2) * 12)%%12))
par(mfrow = c(1, 2))
# Residual from simple linear model
plot(c(time(co2)), resid(linmod <- lm(co2 ~ c(1:n))), ylab = "Residual", main = "Residual\n from linear
    xlab = "Time", pch = 20)
plot(month, resid(linmod), ylab = "Residual", xlab = "Month", main = "Monthly residual\n from linear mo
    pch = 20)
```

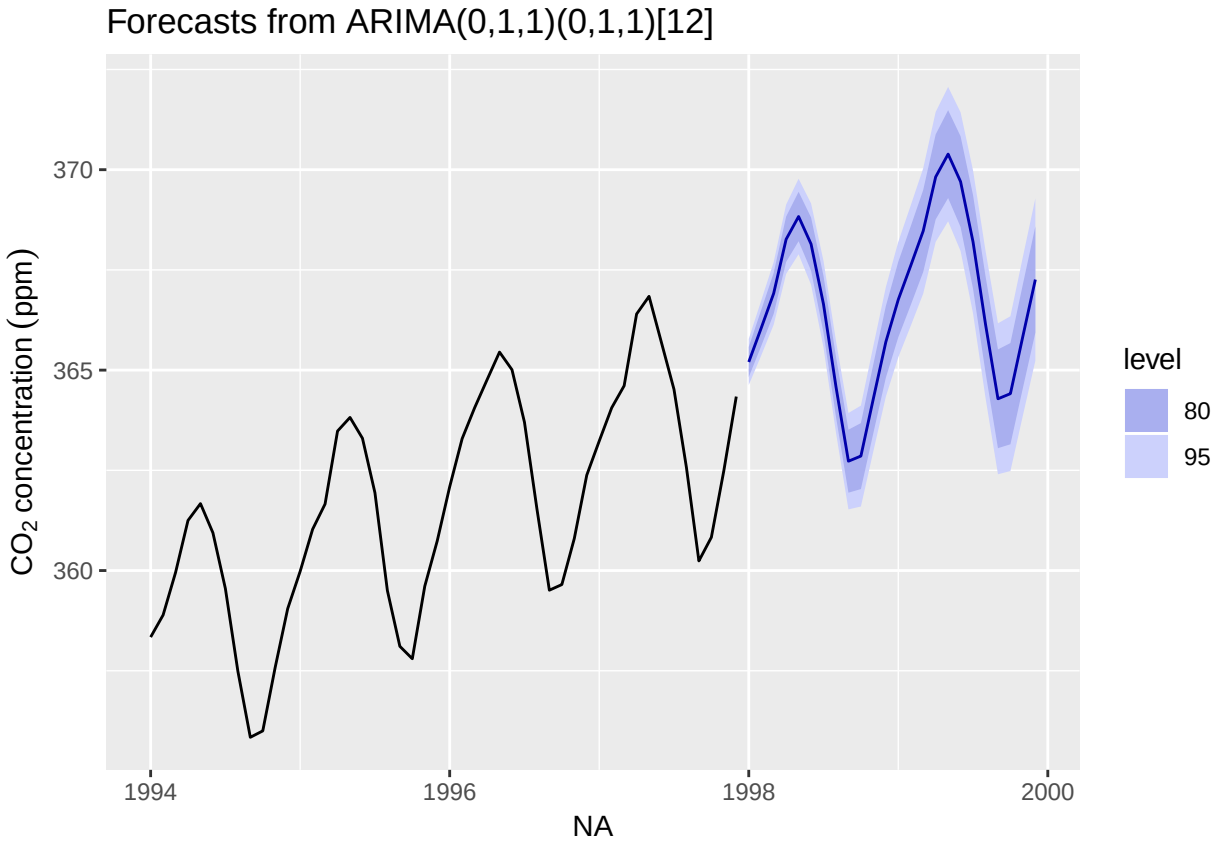
[2] http://robjhyndman.com/hyndsight/arimax/




```
par(mfrow = c(1, 1))
plot(diff(diff(co2, lag = 12)), ylab = expression(CO[2] ~ differences ~ at ~
    lag ~ 1 ~ and ~ 12 ~ (ppm)), bty = "l")
```



```
# Forecasting methods illustrated Commented methods are the default
library(forecast)
# Selected model - see data analysis handout by Prof. Davison
tsmod_co2 <- Arima(co2, order = c(0, 1, 1), seasonal = c(0, 1, 1))  #Fit an ARIMA model
# tsmod_co2 <- arima(co2, order = c(0, 1, 1), seasonal = list(order = c(0,
# 1, 1), period = 12))
for_std <- forecast(tsmod_co2)  #returns matrix with forecast, 80 and 95% pointwise confidence interval
# for_std <- predict(tsmod_co2, n.ahead = 24) # To get equivalent output to
# that of the forecast package, can derive confint manually viz
# cbind(for_std$pred, for_std$pred-qnorm(0.975)*for_std$se,
# for_std$pred+qnorm(0.975)*for_std$se)
yl <- expression(CO[2] ~ concentration ~ (ppm))  #label for plot
autoplot(for_std, include = 48, ylab = yl)
```

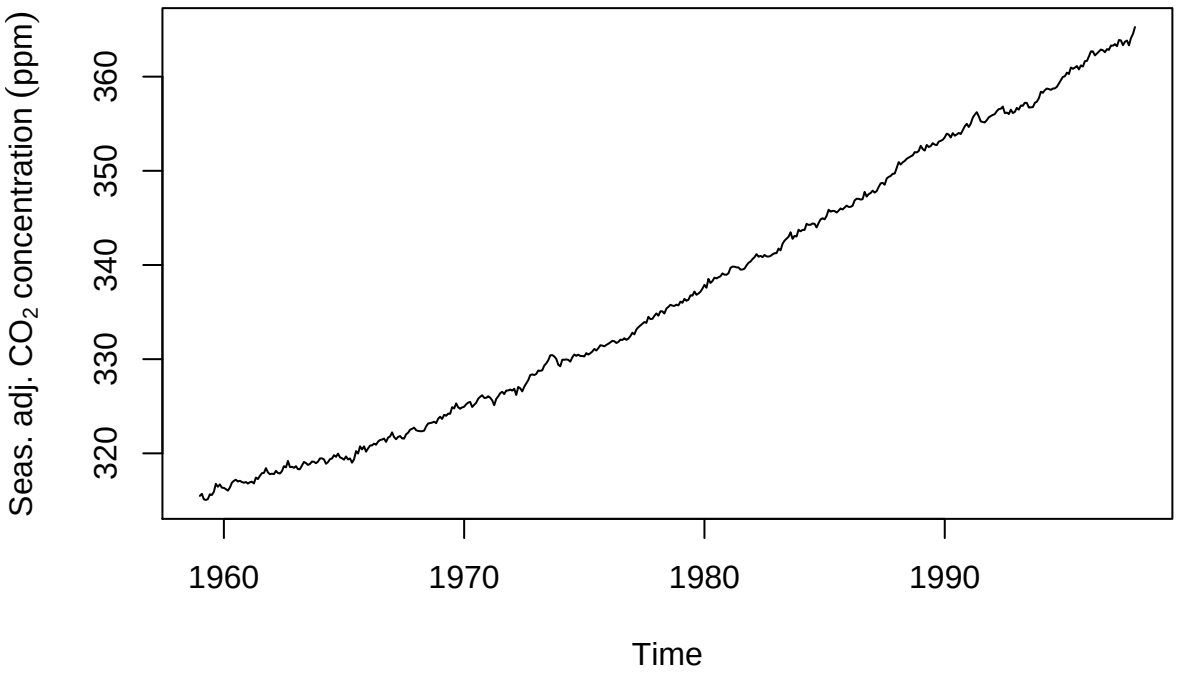



```
# ggplot2 plot, focusing on last four years for readability + two predicted
# plot(for_std, include = 48, ylab = yl) #regular R plot
```

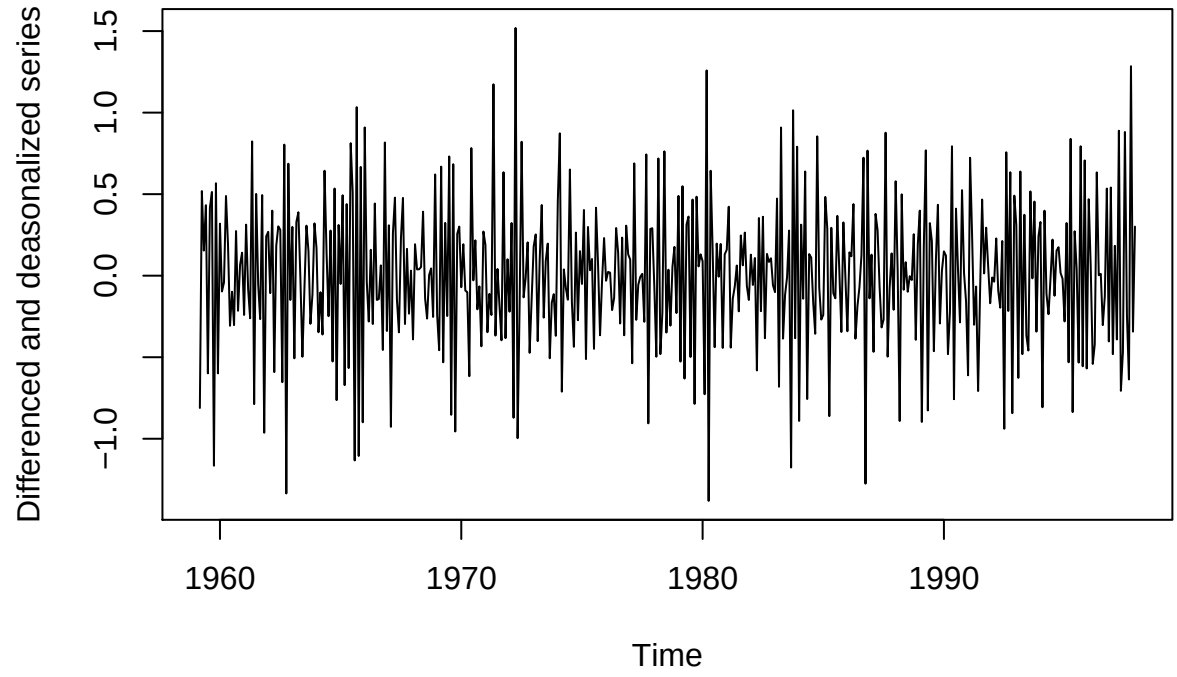
Contrast this prediction with one that includes a drift linear drift, but without the unit root (set `include.drift = TRUE`). You can also try fitting a deterministic seasonal variation, removing the latter, fitting a SARIMA model to the deseasonalized series and including back the cyclic seasonal component in the forecast. Here is an example of how to proceed:

```
# Fit a seasonal + trend decomposition to data, with robust LOESS
stl_co2 <- stl(co2, s.window = "periodic", t.window = 99, robust = TRUE)
# Extract and plot the seasonally adjusted component
```

```
deseas_co2 <- seasadj(stl_co2)
plot(deseas_co2, ylab = expression(Seas. ~ adj. ~ CO[2] ~ concentration ~ (ppm)))
```

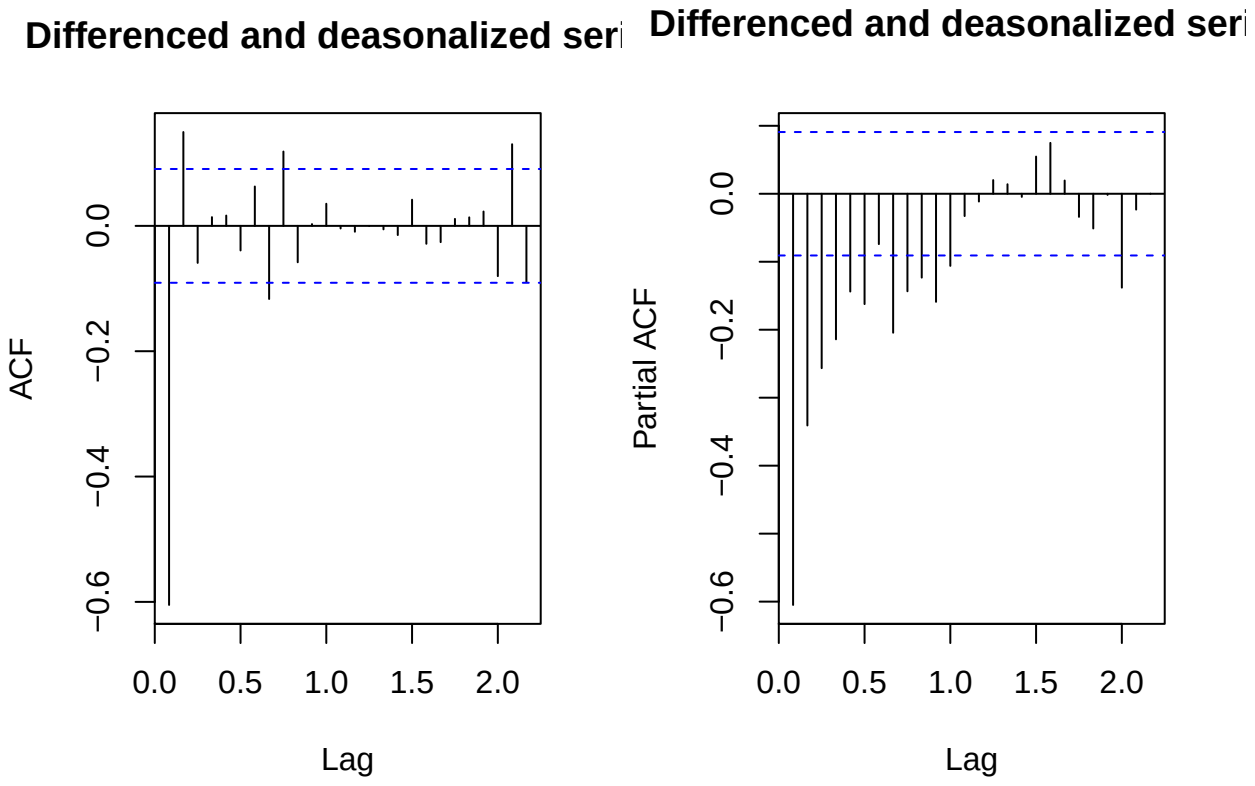



```
# Fit a seasonal model, with double differencing to remove trend and level
plot(deseas_diff_co2 <- diff(diff(deseas_co2)), ylab = "Differenced and deasonalized series")
```

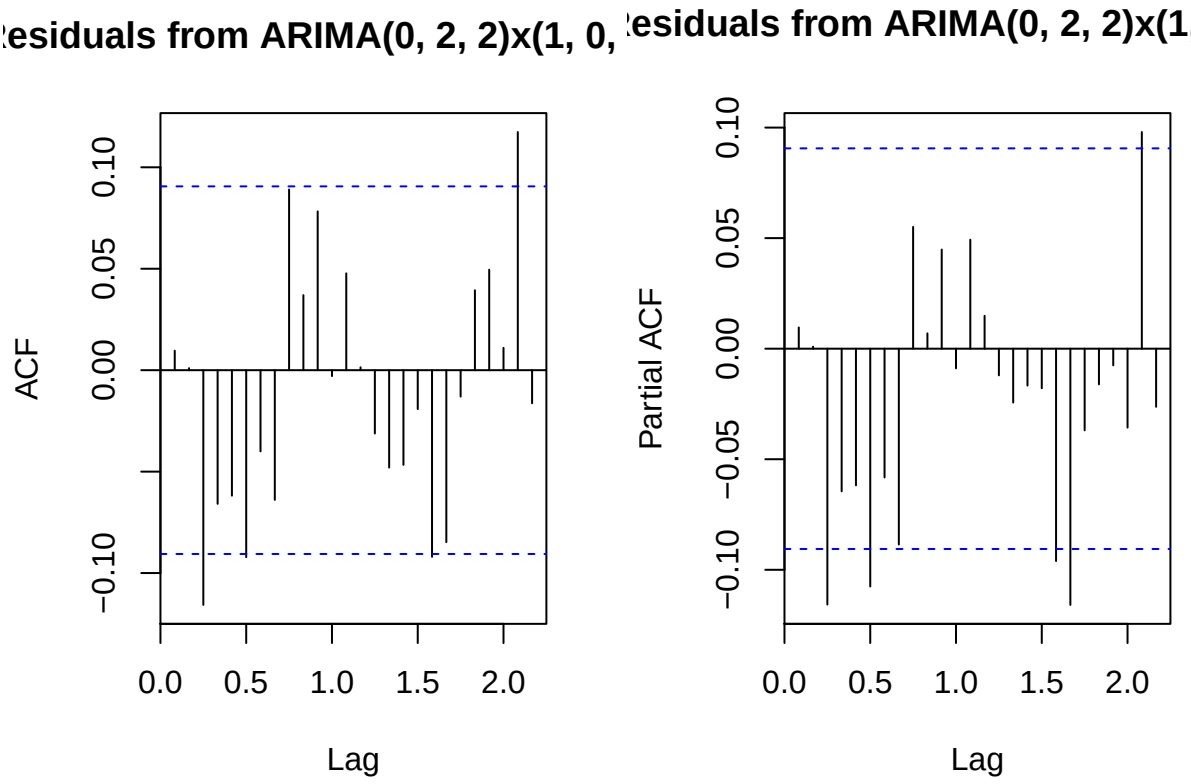



```
# Box-Jenkins model selection
par(mfrow = c(1, 2))
```

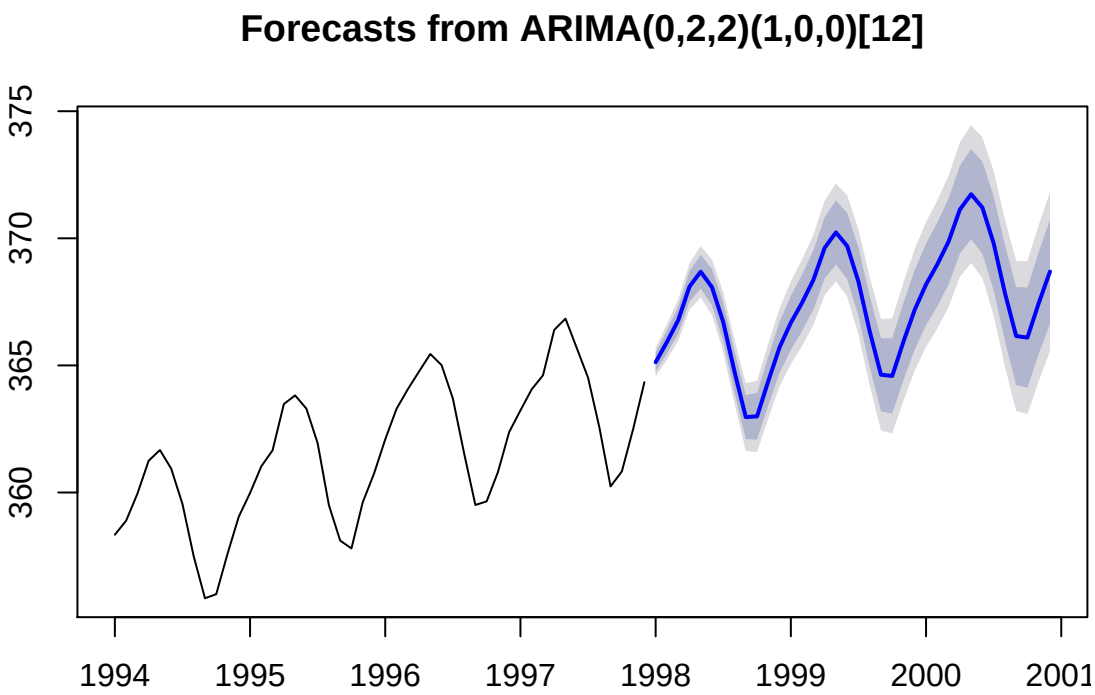
```
TSA::acf(deseas_diff_co2, main = "Differenced and deasonalized series")
pacf(deseas_diff_co2, main = "Differenced and deasonalized series")
```




```
# Trial and error Suggests an MA(2) + autoregressive part (seasonal) b/c not
# purely deterministic!
deseas_co2_arima <- Arima(deseas_co2, order = c(0, 2, 2), seasonal = c(1, 0,
    0))
# (Partial) correlogram of residuals?
main <- "Residuals from ARIMA(0, 2, 2)x(1, 0, 0)[12]"
TSA::acf(resid(deseas_co2_arima), main = main)
pacf(resid(deseas_co2_arima), main = main)
```




```r
# Omitted step: Ljung-Box test for ARIMA residuals Forecasting three years
# ahead using the deseasonalized series
for_stl <- forecast(deseas_co2_arima, h = 36)
# Adding back the seasonal component
for_stl$mean <- for_stl$mean + seasonal(stl_co2)[1:36]
for_stl$lower <- for_stl$lower + seasonal(stl_co2)[1:36]
for_stl$upper <- for_stl$upper + seasonal(stl_co2)[1:36]
# Substituting the deseasonalized series by the original series
for_stl$x <- co2
# Plot the forecast with four previous observed year as guideline
par(mfrow = c(1, 1))
plot(for_stl, include = 48)
```



In practice, it seems clear that the uncertainty is underestimated. We did look at multiple models (or would normally have) before settling on this particular SARIMA model. The parameter uncertainty also needs to be considered, as well as prior data transformation. In the sequel, we provide a brief overview of boostrap methods for time series.

3.1.2 Exercice 1: Nottingham average monthly temperature and Hong Kong monthly exports

1. Fit a SARIMA model on the Nottingham average monthly temperature dataset `nottem` and obtain predictions for the three subsequent years.
2. Fit a SARIMA model to the latter, this time including seasonal dummies as regressors
3. Compare the forecasts from both models
4. Import the dataset `hk_trade_m.csv` using the command `HKTrade<-read.csv("http://sma.epfl.ch/~lbelzile/math34` `header = FALSE)` and try fitting a suitable SARIMA model to the exports (on the log scale). Contrast the forecasts for different models with their complexity as described by the number of components. What do you notice? Plot the forecasted values on the original scale of the data after back-transforming.

3.2 Boostrap methods for time series

The boostrap is a computer-intensive resampling-based methodology that arises as alternative to asymptotic theory.

The idea of the bootstrap is to approximate the data generating process. Suppose our time series $\boldsymbol{Y} = \{Y_1, \ldots, Y_T\}$ is generated by some model DGP. We approximate the latter by an estimate $\widehat{\text{DGP}}$ and use this model to simulate new replicate series $\boldsymbol{Y}^* = \{Y_1^*, \ldots, Y_n^*\}$. We can then reproduce the estimation of the quantity of interest, say $\widehat{\boldsymbol{\theta}}$, by repeating the estimation procedure over the (new) simulated datasets. The relationship between the test statistic and the population value $\widehat{\boldsymbol{\theta}} - \boldsymbol{\theta}$ should be also closely approximated by the bootstrap replicates $\widehat{\boldsymbol{\theta}}^* - \boldsymbol{\theta}^*$, so it is paramount that the latter reproduce the features under study.

To recap: rather than relying on the asymptotic distribution of the test statistics, one simulates artificial datasets

under the postulated model and calculate the test statistic on each replicate dataset. For testing, this model should correspond to sampling under the *null hypothesis*. We use the empirical distribution of the so-called B bootstrap replicates as distribution for the test statistic to calculate standard errors, confidence intervals, critical values or P-values.

We illustrate the use of the boostrap on a simple example from linear models, than detail its use in time series.

3.2.1 Bootstrapping a linear model

Consider a simple linear model with independent and identically distributed errors,

$$\boldsymbol{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where $\boldsymbol{\varepsilon} \overset{\text{iid}}{\sim} F(\mathbf{0}, \sigma^2\mathbf{I}_n)$ and the first column of the $n \times k$ matrix of regressors $\mathbf{X}$ is $\mathbf{1}_n$. The parameter estimated obtained by ordinary least squares are such that the fitted values are orthogonal to the estimated errors, meaning $\mathbf{X}\hat{\boldsymbol{\beta}} \perp \hat{\boldsymbol{\varepsilon}}$ by construction, and that $\bar{\hat{\boldsymbol{\varepsilon}}} = 0$ provided we include $\mathbf{1}_n$ as a regressor.

Suppose for simplicity that the linear model takes the form $y_i = \alpha + \beta x_i + \varepsilon_i$, where $\varepsilon_i \overset{\text{iid}}{\sim} F(0, \sigma^2)$ and that we want to test the null hypothesis $\mathsf{H}_0 : \beta = 0$.

The simplest bootstrap scheme is the nonparametric bootstrap, due to Efron (1976). It goes as follows: the bootstrap data generating process consists in resampling residuals (with replacement) from $\tilde{F}$, the empirical distribution of the errors.

1. Fit the model $\boldsymbol{y} = \widehat{\alpha} + \widehat{\beta}\boldsymbol{x} + \widehat{\boldsymbol{\varepsilon}}$; the Wald test statistic takes the form $T = \hat{\beta}/\text{se}(\hat{\beta})$.
2. Estimate the model postulated under H_0 and obtain residuals $\widetilde{\boldsymbol{\varepsilon}} = \boldsymbol{y} - \widetilde{\alpha}$.
3. Create bootstrap series $\boldsymbol{y}_b^* = \widetilde{\alpha}\mathbf{1}_n + \boldsymbol{\varepsilon}_b^*$ for $b = 1, \ldots, B$, where $\varepsilon_{i_b}^*$ are resampled with replacement from the empirical distribution $\{\widetilde{\varepsilon}\}_{i=1}^n$ with probability $1/n$.
4. Obtain bootstrap test statistics by fitting $\boldsymbol{y}_b^* = \widehat{\alpha}_b^* + \widehat{\beta}_b\boldsymbol{x} + \widehat{\boldsymbol{\varepsilon}}_b^*$ and obtain $T_b = \hat{\beta}_b/\text{se}(\hat{\beta}_b)$
5. The P-value for T will be the rescaled rank

$$\frac{1}{B}\sum_{b=1}^{B} \mathrm{I}(T_b > T)$$

for a one sided test, otherwise we use the rank of the absolute values, $\sum_{b=1}^{B} \mathrm{I}(|T_b| > |T|)$ for a two-sided test.

Nothing prevents one to take $T = \hat{\beta}$ as test statistic in the above. It is however best to bootstrap a pivotal quantity should the latter be available. Indeed, if the test statistic of interest is pivotal under the null hypothesis, then the bootstrap is a Monte-Carlo test and the latter is exact at level α if $\alpha(B+1)$ is integer.

The parametric bootstrap for the linear regression would specify a model for the generic distribution F, for example Normal. One construct bootstrap series as before, this time replacing the sampling in Step 3. with $\boldsymbol{\varepsilon}_b^* \sim \mathcal{N}_n(\mathbf{0}_n, \widehat{\sigma}^2\mathbf{I}_n)$, where the estimator $\widehat{\sigma}^2$ of the model fitted under H_0 should be the unbiased estimator of the variance, whose denominator is $n-k-1$ rather than n.

Many test statistics can be cast as regression problems. We have seen that conditional maximum likelihood estimates for AR(p) models coincide with least square estimates from the regression $y_t = \mu + \boldsymbol{\phi}^\top(y_{t-1}, \ldots, y_{t-p})^\top$ for $t = p, \ldots, n$ because of the Markov structure.

3.2.2 Testing for heteroscedasticity

Consider a GARCH(1,1) model $y_t = \mu + v_t$, $v_t = \sigma_t\varepsilon_t, \varepsilon_t \overset{\text{iid}}{\sim} F(0, \sigma^2)$ and

$$\sigma_t^2 = \alpha_0 + \alpha_1 v_{t-1}^2 + \beta\sigma_{t-1}^2$$

and suppose that we wish to test the null hypothesis that the $v_t \overset{\text{iid}}{\sim} F(0,\sigma^2)$. The easiest way to test this would be to run the regression $\widehat{v}_t^2 = b_0 + b_1 \widehat{v}_{t-1}^2 + \eta_t$, where $\widehat{v}_t$ are residuals from a linear model $y_t = \mu + v_t$. The null hypothesis that $\alpha_1 = \beta = 0 = 0$ is equivalent to testing $b_1 = 0$, and the ordinary t statistic could be used and we are back in the framework just presented.

Bootstrap replicates must reproduce the postulated model. This is complicated for time series, because of the serial dependence and the potential heteroscedasticity. If we resample residuals without taking into account the time dependence, our replicates won't be anymore time series!

3.2.3 AR-sieve bootstrap

Approximating a stationary time series by an AR(p) leads to a parametric bootstrap termed "sieve" bootstrap, after Grenander (1981). Under what kind of assumptions can AR models reproduce features of the underlying stochastic process? By the Wold-decomposition, any purely stochastic mean-zero stationary process with a positive and continuous spectral density can be written as

$$Y_t = \sum_{j=1}^{\infty} \phi_j Y_{t-j} + \varepsilon_t.$$

for uncorrelated white noise sequence ε_t. Another possible (stronger) assumption is to assume that the process Y_t admits an AR(∞) representation,

$$Y_t - \mu = \sum_{j=1}^{\infty} \varphi_j (Y_{t-j} - \mu) + e_t, \qquad e_t \overset{\text{iid}}{\sim} F(0,\sigma^2), \quad \sum_{j=1}^{\infty} |\varphi_j| < \infty.$$

Under the bootstrap scheme, the replicated white noise series will have the same marginal properties as those of the original ε_t. The model is Markov, so we can simulate the observations in a recursive fashion. We proceed as follows

1. Estimate residuals $\widehat{\varepsilon}_t = \sum_{j=0}^{p} \widehat{\phi}_j (X_{t-j} - \widehat{\mu})$ for $t = p+1, \ldots, n$.
2. Center the residuals $\widetilde{\varepsilon}_t = \widehat{\varepsilon}_t - \overline{\widehat{\varepsilon}} = \widehat{\varepsilon}_t - (n-p)^{-1} \sum_{j=p+1}^{n} \widehat{\varepsilon}_j$ for $t = p+1, \ldots, n$.
3. Resample iid realizations ε_t^* from the empirical distribution function of $\{\widetilde{\varepsilon}_t\}$.
4. Simulate $\{Y_t^*\}_{t\in\mathbb{Z}}$ recursively as $Y_t^* = \widehat{\mu} + \sum_{j=1}^{p} \widehat{\phi}_j Y_{t-j}^* + \varepsilon_t^*$. One can initialize with observations and burn-in the Markov chain where necessary.

Remark: when we fit an AR(p) model in bootstrap loops, it is customary to use the Yule–Walker equations since this ensures a stationary and causal solution, does not require optimization and is very fast in contrast to maximum likelihood estimation. It is however wrong to use the latter to estimate AIC or the log-likelihood value!

In the slides, we are interested in confidence intervals for h-step ahead forecasts. The latter are defined as $y_{t+1}^t \pm \mathfrak{z}_{1-\alpha/2}\text{se}(y_{t+1}^t)$ and thus depend on the forecast error $y_{t+1}^t - y_{t+1}$ through estimates of the standard error. However, the latter cannot be estimated with a single realization, so is approximated under the assumption of the parametric model with $\varepsilon_i \overset{\text{iid}}{\sim} F(0,\widehat{\sigma}^2)$. This plug-in method ignores the model selection procedure and the parameter estimation uncertainty.

Estimation procedure: we take data from 1930-1988 and fit an AR model to data from 1930-1979, keeping observations from 1980-1988 to validate our forecasts. We select the AR model whose order minimizes AIC.

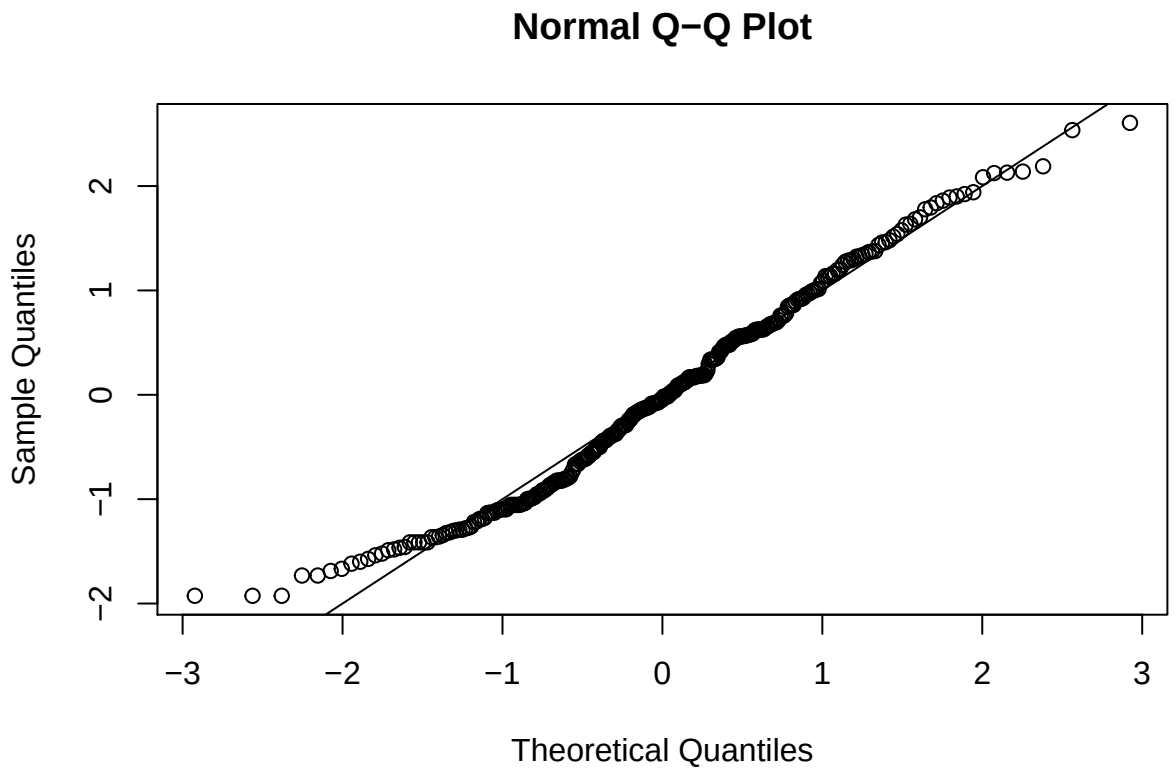
Simulation procedure: The process is irreversible, so we should do forward simulations conditional on observed samples, and it is thus convenient to use some of the data for the period 1700-1930. We simulate from an AR(p) model of high order conditional on p initial values $X_{1930-1}, \ldots, X_{1930-p}$. We then repeat our estimation procedure, fit an AR model to the observations corresponding to the period 1930-1979 in the bootstrap series and obtain h-step ahead forecasts for the years 1980-1988, $X_{t+h}^{*t}, h = 1, \ldots, 9$, that can be compared to the actual realization X_{t+h}^*. For an autoregressive models, the residuals $\{e_t\}$ are the difference between fitted values (the one-step ahead prediction) and observed values, namely $\{X_t^{t-1} - X_t\}$ — this is easily seen if we write the model as a linear regression.

We now detail the bootstrap procedure employed by Prof. Davison in his slides. The sieve bootstrap is employed to account for the variation due to (a) future innovations (b) parameter estimation and (c) model selection. The test statistics are the pair (AIC, prediction errors). The counts are not normally distributed and are overdispersed, with a clear multiplicative structure for the variance and a 11 year cycle. The first simulation uses the AR model that minimizes AIC, while the second simulation ignores model selection uncertainty by fixing the order to 11.

3.2.4 Boostrap models for uncertainty assesment

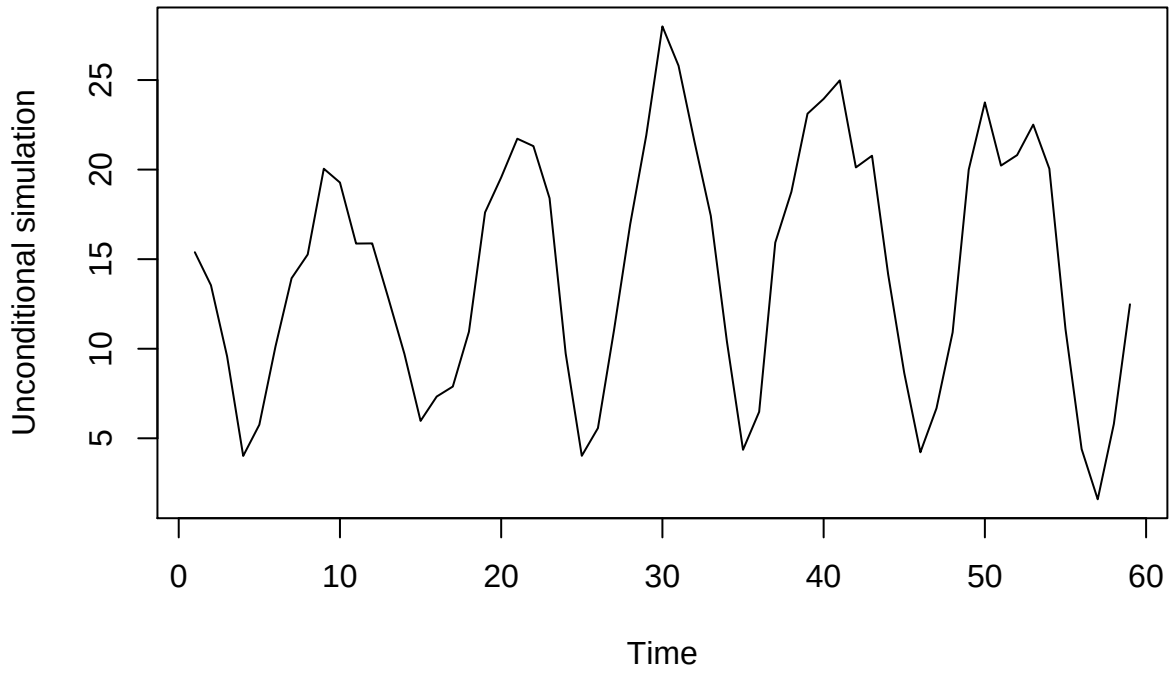
The following code is my own adaptation of Example 8.3 from Davison and Hinkley (1997).

```
#Counts are overdispersed
library(boot); library(forecast)
#Variance stabilizing transform
sun <- 2*(sqrt(sunspot.year+1)-1)
#QQ plot of the variance-transformed observations
qqnorm(scale(sun), pty="s"); abline(a = 0, b = 1)
```



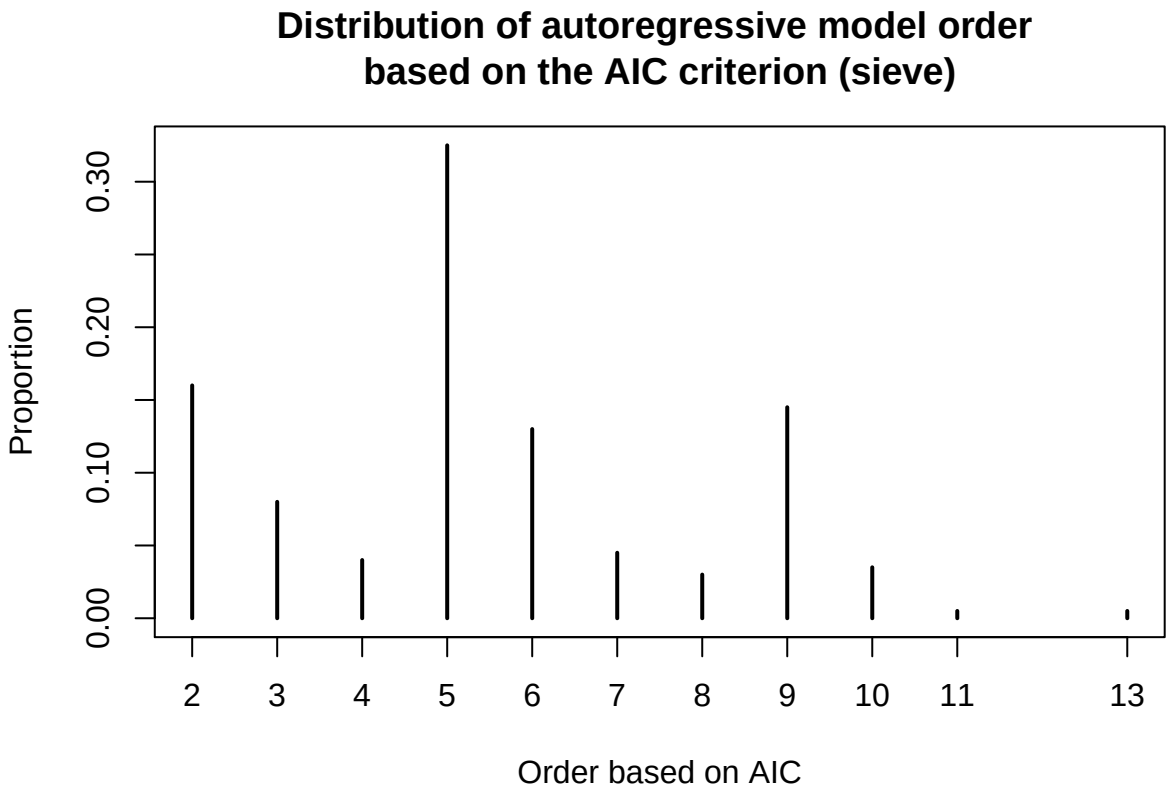

```
#Alternative would be a Box-Cox transform
# sun_bc <- BoxCox(window(sunspot.year, 1818),
#                  forecast::BoxCox.lambda(window(sunspot.year, 1818), method = "loglik"))
# qqnorm(scale(sun_bc)); abline(a = 0, b = 1)
# plot(sun_bc, main = "Average number of sunspots\n(Box-Cox transformed)", ylab = "")
#apparent cycle of 11

#Fit a time series model to transformed sunspot
sun_ar_auto <- forecast::auto.arima(window(sun, 1930, 1979), max.q = 0, max.Q = 0,
                                    max.d = 0, allowdrift = FALSE, max.D = 0, max.P = 0,
```

```
                                    max.p = 25, max.order = 25, stepwise = FALSE, ic = "aic")
p <- sun_ar_auto$arma[1]
res <- sun_ar_auto$residuals #residuals
res <- res - mean(res) #under the null, epsilon are mean zero

ar_coef <- sun_ar_auto$model$phi #coefficients
#Create a list with model components for arima.sim


#Simulate series, resampling errors (should be centered)
#and condition on p preceding values (since they are known, but not used)
sim_unc <- function(tseries, mod, nobs, res){
  init <- rev(window(tseries, c(tail(time(sun),1)-nobs-mod$arma[1]+1),
                     c(tail(time(sun),1)) - nobs)) - mod$coef["intercept"]
  mod$coef["intercept"] + filter(sample(res, size = 59, replace=TRUE),
                   filter = mod$model$phi, method = "recursive", sides = 1,
                   init = init)
}

plot(sim_unc(sun, mod = sun_ar_auto, nobs = 59, res = res), ylab = "Unconditional simulation")
```




```
#Boostrap statistics, returned as a list here
boot_stat <- function(tseries, npredict, p, fixorder = FALSE){
  n <- length(tseries)
  #Fit the AR model
  if(fixorder){
    ar_boot <- try(forecast::Arima(tseries[-((n-npredict+1):n)], order = c(p, 0, 0),
                                   include.mean = TRUE, include.drift = FALSE, method = "ML"))
    if(is.character(ar_boot)){
      return(list(forecast_error = rep(NA, npredict),  #forecast error
            ar_order = p, #order of AR component
```

```
            mu = NA #intercept
            )
        )
    }
  } else {
    ar_boot <- forecast::auto.arima(tseries[-((n-npredict+1):n)],
                                max.q = 0, max.Q = 0, max.d = 0, allowdrift = FALSE,
                                max.D = 0, max.P = 0, max.p = 25,
                                max.order = 25, stepwise = FALSE)
  }
  #Obtain forecast error for 9 periods ahead (equivalent of 1980-1988) for simulated data
  for_err <- as.vector(tseries[(n-npredict+1):n] - forecast(ar_boot, h = npredict)$mean)

#Collect test statistics
return(list(forecast_error = for_err,  #forecast error
            ar_order = ar_boot$arma[1], #order of AR component
            mu = ar_boot$coef["intercept"] #intercept
            )
       )
}


boot_full <- replicate(n = 199, expr = boot_stat(sim_unc(tseries = sun,
                              mod = sun_ar_auto, res = res, nobs = 59),
                              npredict = 9, p = p))

boot_fixed <- replicate(n = 199, expr = boot_stat(sim_unc(tseries = sun,
                              mod = sun_ar_auto, res = res, nobs = 59),
                              npredict = 9, p = p, fixorder = TRUE))



#bootstrap replicates - obtain the standard errors of the forecast errors
for_err_full_boot <- apply(t(matrix(unlist(boot_full[1,]), nrow = 9)), 2, sd) #unconditional
for_err_fixed_boot <- apply(t(matrix(unlist(boot_fixed[1,]), nrow = 9)), 2, sd, na.rm = TRUE) #AR(11)
#AR order
ar_order_boot <- unlist(boot_full[2,])
plot(table(c(sun_ar_auto$arma[1], ar_order_boot))/(1+length(ar_order_boot)), ylab = "Proportion", xlab =
      main = "Distribution of autoregressive model order \nbased on the AIC criterion (sieve)")
```




```
forec <- forecast(sun_ar_auto, h = 9)
#Forecasts from forecast::forecast does not return the se
#So reverse-engineer the calculation to retrieve those
forec$se <- (-forec$lower[, 1] + forec$mean)/qnorm(0.5 * (1 + forec$level[1]/100))
```

```
library(knitr)
tab <- rbind(c(forec$se), for_err_full_boot, for_err_fixed_boot)
row.names(tab) <- c("Nominal", "AR", "AR(11)")
colnames(tab) <- as.character(1:9)
kable(tab, caption = "h-step ahead prediction standard errors", digits = 2)
```

\begin{table}[t]

\caption{(#tab:prediction_error_table)h-step ahead prediction standard errors}

	1	2	3	4	5	6	7	8	9
Nominal	2.09	3.18	3.60	3.60	3.61	3.66	3.73	3.76	3.81
AR	2.19	3.27	3.89	4.12	4.19	4.08	3.69	4.01	3.97
AR(11)	2.04	3.28	3.63	3.74	3.73	3.75	3.63	3.87	3.91

\end{table}

While I have computed manually everything, the package `boot` provides options for bootstrap and wrappers for standard methods. The sieve bootstrap is implemented using `tsboot` with option `sim = model`, reflecting the fact that it is model-based. The example in the slides (and from the book *Bootstrap methods and their applications*, example 8.3, used the `ar` command to fit the autoregressive model via the Yule-Walker equations). The function `ar` subtract the sample mean to ensure the errors are residuals are centered and uses a conditional model. As mentioned before, don't use the latter to estimate an information criterion (although it is okay for predictions).

```
library(boot)
# Estimate the AR coefficients
sun_ar <- ar(window(sun, 1930, 1979), aic = FALSE, order.max = p)
# ar automatically selects order by AIC unless `aic = FALSE` in which case
# it fits the model with order.max
sun_ar$order
```

```
[1] 11
```

```
model <- list(ar = sun_ar$ar, order = c(p, 0, 0))
# Statistic under study with the bootstrap Manual fitting and collection of
# the results
sun_fun <- function(tsb) {
    ar.fit <- ar(window(tsb, 1, 50), aic = FALSE, order.max = p)
    # Fitted using Yule-Walker equations, to avoid convergence issues and
    # because it is MUCH faster
    c(mean(tsb), c(predict(ar.fit, newdata = window(tsb, 1, 50), n.ahead = 9,
        se.fit = FALSE) - window(tsb, 51, 59)))
    # return prediction of time series, mean
}

# Simulation from fitted AR model, with arguments res: residuals from model
# fit n.sim: length of series to simulate ran.args: list with components
# `ar` and `order` From 'Bootstrap methods and their applications',
# copyright CUP ran.gen must have precisely these arguments, in this order.
sun_sim <- function(tseries, n.sim, ran.args) {
    rg1 <- function(n, res) {
        sample(res - mean(res), n, replace = TRUE)
    }
    ts.orig <- ran.args$ts
    ts.mod <- ran.args$model
    mean(ts.orig) + ts(arima.sim(model = ts.mod, n = n.sim, rand.gen = rg1,
        res = as.vector(ran.args$res)))
}
# Model based bootstrap Specify the ARIMA model parameters
sun_model <- list(order = c(sun_ar$order, 0, 0), ar = sun_ar$ar)
sun_res <- c(scale(sun_ar$resid[!is.na(sun_ar$resid)], scale = FALSE))
# Sieve bootstrap - also computes the test statistic on the original dataset
# hence problems, because would usually pass residuals, and these are
# shorter and have different time stamps use orig.t = FALSE to desactivate
# this option
sun_boot <- tsboot(ts(c(window(sun, 1930))), sun_fun, R = 999, sim = "model",
    n.sim = 59, ran.gen = sun_sim, ran.args = list(res = sun_res, ts = window(sun,
        1930), model = sun_model))

# Standard deviations of prediction error
apply(sun_boot$t[, -1], 2, sd)
```

```
[1] 2.500653 3.559976 3.789827 3.687904 3.661806 3.609122 3.735139 3.867332
[9] 3.838362
```

More details about bootstrap methods for time series can be found in Bühlmann (2002)[3] or Kreiss and Lahiri

[3] https://projecteuclid.org/euclid.ss/1023798998

(2012)[4].

More sophisticated methods will be needed depending on the model considered. If you have heteroscedasticity, then the residuals v_t cannot be sampled independently from the fitted residuals. One can use then the so-called **wild** bootstrap and resample $v_t^* = s_t^* v_t$, where $s_t^* \overset{\text{iid}}{\sim} F(0,1)$. The simplest such example is the Rademacher distribution, a two point distribution that puts mass 1/2 on $\{-1,1\}$. One could also resample from $\mathcal{N}(0,1)$.

3.2.5 Exercice 2: Lake Erie and Lake Huron levels

1. Fit a linear model to the January observations of the Lake Erie data of the form $y_t = \beta_0 + \beta_1 t + \varepsilon_t$, ignoring the temporal dependence. Test the null hypothesis that the trend is not significant.
2. Use a parametric sieve bootstrap with normal errors to assess the presence of a trend in the Lake Huron dataset. Report your conclusion as a P-value.
3. Recall the estimation of the Lake Huron level in Practical 2. There, we saw that fitting an ARMA(2,1) led to a parameter value of $\theta_1 \approx 1$. Using a parametric bootstrap, test the hypothesis that the parameter $\theta_1 = 0$. (Indication: fit an AR(2) model, simulate replicates of the latter and fit an ARMA(2,1) model to the bootstrap time series. Compare the value of your test statistic $\hat{\theta}_1$ with the estimates $\{\hat{\theta}_{1b}^*\}_{b=1}^B$).

3.3 Generalized Autoregressive Conditional Heteroskedasticity (GARCH) models and extensions

We look at estimation of (G)ARCH models. The simple ARMA(p,q)-GARCH(m,r) model is

$$Y_t - \mu = \sum_{h=1}^{p} \phi_h (Y_{t-h} - \mu) + \sum_{l=1}^{q} \vartheta_l V_{t-l} + V_t, \qquad V_t = \sigma_t \varepsilon_t, \quad \varepsilon_t \overset{\text{iid}}{\sim} F(0,1)$$
$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{m} \alpha_i V_{t-i}^2 + \sum_{j=1}^{r} \beta_j \sigma_{t-j}^2.$$

The residuals from the ARMA models, V_t thus follow a GARCH model. The parameters of the GARCH $\alpha_0, \alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_r$ must be positive. Furthermore, for the variance σ_t^2 to be stationary, we require $\sum_{i=1}^m \alpha_i + \sum_{j=1}^r \beta_j < 1$, akin to the ARMA setting. The V_t are typically leptokurtic (and thus heavy-tailed) as a result of the conditional heteroscedastic structure.

If we have $V_t = \sigma_t^2 \varepsilon_t$, where $\varepsilon_t \overset{\text{iid}}{\sim} F(0,1)$ is iid white noise, then

$$\mathrm{E}(V_t^2 \mid \mathcal{H}_{t-1}) = \sigma_t^2.$$

Here, $\mathcal{H}_{t-1}$ denotes the information set at time $t-1$. One can see that V_t^2 has the form of an ARMA(m,r) process, but the iid white noise is replaced with a martingale difference sequence η_t. We can rewrite

$$V_t^2 - \nu = \sum_{i=1}^{\max\{m,r\}} (\alpha_i + \beta_i)(V_{t-i}^2 - \nu) - \sum_{j=1}^{r} \beta_j \eta_{t-j} + \eta_t,$$

where $\nu = \alpha_0 / \sum_{i=1}^{\max\{m,r\}} (\alpha_i + \beta_i)$, and we define for simplicity $\alpha_i = 0$ for $i > m$ and β_j for $j > r$. We can identify from there the AR coefficients $\tilde{\phi}_i = \alpha_i + \beta_i$ and the MA coefficients $\tilde{\theta}_j = -\beta_j$. A necessary condition for covariance stationarity is that $\sum_{i=1}^m \alpha_i + \sum_{j=1}^r \beta_j < 1$, The strategy for modelling is thus to fit an ARIMA-type model to the original series, then examine squared residuals. One suitable orders p, q, m, r have been determined, we can estimate the models jointly. The autocorrelation function of the GARCH(1,1),

[4] https://www.kevinsheppard.com/images/0/0a/Kreiss_and_lahiri.pdf

$$\rho(1) = \frac{\alpha_1(1-\beta_1^2-\alpha_1\beta_1)}{(1-\beta_1^2-2\alpha_1\beta_1)}$$
$$\rho(h) = (\alpha_1+\beta_1)\rho(h-1), \quad h \geq 2,$$

decays geometrically at rate $\alpha_1 + \beta_1$.

Joint estimation of ARMA-GARCH type models can be handled with functions from the `rugarch` package. Apart from the documentation of the package, there is a worked out example here[5] and more examples on the package author's blog[6]. You can specify different types of GARCH model other than the standard GARCH, or `sGARCH`, add external regressors and choose the distribution function of the errors. This provides a neat way to include an ARMA-GARCH type model for your analysis.

How does one proceed with the estimation of a GARCH model? Maximum likelihood is the standard option, but the MLE must be found numerically. This function from a preprint by Würtz, Chalabi and Luskan[7], shows how to construct the likelihood for a simple GARCH(1,1) model.

```
# From Wurtz, Chalabi, Luskan (JSS)
garch11Fit = function(x) {
    require(numDeriv)
    # Step 1: Initialize Model Parameters and Bounds:
    Mean = mean(x)
    Var = var(x)
    S = 1e-06
    params = c(mu = Mean, omega = 0.1 * Var, alpha = 0.1, beta = 0.8)
    lowerBounds = c(mu = -10 * abs(Mean), omega = S^2, alpha = S, beta = S)
    upperBounds = c(mu = 10 * abs(Mean), omega = 100 * Var, alpha = 1 - S, beta = 1 -
        S)
    # Step 2: Set Conditional Distribution Function:
    garchDist = function(z, hh) {
        dnorm(x = z/hh)/hh
    }
    # Step 3: Compose log-Likelihood Function:
    garchLLH = function(parm) {
        mu = parm[1]
        omega = parm[2]
        alpha = parm[3]
        beta = parm[4]
        z = (x - mu)
        Mean = mean(z^2)
        # Use Filter Representation:
        e = omega + alpha * c(Mean, z[-length(x)]^2)
        h = filter(e, beta, "r", init = Mean)
        hh = sqrt(abs(h))
        -sum(log(garchDist(z, hh)))  #llh

    }
    # print(garchLLH(params)) Step 4: Estimate Parameters and Compute
    # Numerically Hessian:
    fit = nlminb(start = params, objective = garchLLH, lower = lowerBounds,
```

[5] http://unstarched.net/wp-content/uploads/2013/06/an-example-in-rugarch.pdf
[6] http://unstarched.net/r-examples/rugarch/a-short-introduction-to-the-rugarch-package/
[7] http://www-stat.wharton.upenn.edu/~steele/Courses/956/RResources/GarchAndR/WurtzEtAlGarch.pdf

```
        upper = upperBounds)
    Hessian <- numDeriv::hessian(func = garchLLH, x = fit$par)
    # Step 5: Create and Print Summary Report:
    se.coef = sqrt(diag(solve(Hessian)))
    tval = fit$par/se.coef
    matcoef = cbind(fit$par, se.coef, tval, 2 * (1 - pnorm(abs(tval))))
    dimnames(matcoef) = list(names(tval), c(" Estimate", " Std. Error", " t value",
        "Pr(>|t|)"))
    cat("\nCoefficient(s):\n")
    printCoefmat(matcoef, digits = 6, signif.stars = TRUE)
}
```

We consider in the sequel the log-returns of the Microsoft stock.

```
library(tseries) #to extract quotes from internet
#Obtain Microsoft series from Internet
msft.prices = get.hist.quote(
instrument = "MSFT",
quote = "Close",  #adjusted does not exist anymore
provider = c("yahoo"), origin = "1999-12-30", start="2000-01-01", end="2010-01-01",
retclass = c("zoo"), quiet = FALSE, drop = FALSE)
```

```
time series starts 2000-01-03
time series ends   2009-12-31
```

```
#Transform to log-returns
msft <- as.data.frame(msft.prices)
N <- length(msft[, 1])
msft.returns <- 100*(log(msft[2:N, ])-log(msft[1:(N-1), ]))
```

Now that we have our financial time series, we can try fitting a GARCH model to it. We start with the code provided above.

```
# Fit with function akin to that found in library(fGarch)
garch11_model <- garch11Fit(msft.returns)
```

```
Coefficient(s):
        Estimate  Std. Error  t value   Pr(>|t|)
mu    0.00428664  0.03218567  0.13318    0.89405
omega 0.07139907  0.01408702  5.06843 4.0111e-07 ***
alpha 0.08454825  0.01262800  6.69530 2.1523e-11 ***
beta  0.90354370  0.01320064 68.44697 < 2.22e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
kurtosis_garch11 <- function(alpha1, beta1) {
    3 * (1 + alpha1 + beta1) * (1 - alpha1 - beta1)/(1 - beta1^2 - 2 * alpha1 *
        beta1 - 3 * alpha1^2)
}
kurtosis_garch11(garch11_model[3, 1], garch11_model[4, 1])
```

```
[1] 7.573766
```

The parameter estimates for the parameters α_1, β_1 are typical of those of stock returns (α_1 small, β_1 large, their sum close to 1). The parameter α_0 is the positive constant α_0, which is small. For the process to be covariance stationary (meaning it has finite fourth moments), we need $\alpha_1+\beta_1<1$. The closer the sum $\alpha_1+\beta_1$ to 1, the slower is the decay in the autocorrelation function of squared residuals. Here, the estimates are 0.9880919, which is close to the boundary and is nearly integrated.

The kurtosis of the GARCH(1,1) model is

$$\kappa = \frac{\mathrm{E}\{(Y-\mu)^4\}}{[\mathrm{E}\{(Y-\mu)^2\}]^2} = \frac{3(1+\alpha_1+\beta_1)(1-\alpha_1-\beta_1)}{(1-\beta_1^2-2\alpha_1\beta_1-3\alpha_1^2)}$$

if $3\alpha_1^2+2\alpha_1\beta_1+\beta_1^2<1$, provided the fourth moment exist (when $0\leq\alpha_1+\beta_1<1$) and that $\alpha_1<3^{-1/2}$. Recall for reference that the kurtosis of a standard normal variate is 3. The unconditional variance will be proportional to $\alpha_0/(1-\alpha_1-\beta_1)$. If we specify the errors of the distribution to be Student-t, then the hierarchical construction can be employed to derive the mle.

I will now illustrate the routines in the package `rugarch`, which returns S4 objects. This class has slots (accessible via `@` and methods. See `?uGARCHfit-class` for more info.

```
library(rugarch)
# Specification of GARCH(1, 1) model using rugarch workhorse with ARMA(1,0)
# + mean, normal errors
model <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1,
    1)), mean.model = list(armaOrder = c(1, 0), include.mean = TRUE), distribution.model = "norm")
# Model fitting
model_fit <- ugarchfit(spec = model, data = msft.returns, solver = "nloptr",
    solver.control = list(solver = 9))
# Did the optimization routine converge?
convergence(model_fit)  #0 == TRUE, indicating convergence
```

```
[1] 0
```

A nice feature of this function is the possibility to include regressors in the mean and in the variance. The arguments `variance.model` and `mean.model`, which are lists, can be given `external.regressors` matrices. For the variance, the standard formulation as σ_t^2 is not adequate since we possibly get negative variance with regressors. The choice of an `eGARCH`, which models $\log(\sigma_t^2)$, solves this problem. The eGARCH(m,r) model for $V_t=\sigma_t\varepsilon_t$ is

$$\log(\sigma_t^2)=\alpha_0+\sum_{i=1}^{m}\gamma_i\{|\varepsilon_{t-1}|-\mathrm{E}(|\varepsilon_{t-1}|)\}+\alpha_i\varepsilon_{t-1}+\sum_{j=1}^{r}\beta_j\log(\sigma_{t-1}^2).$$

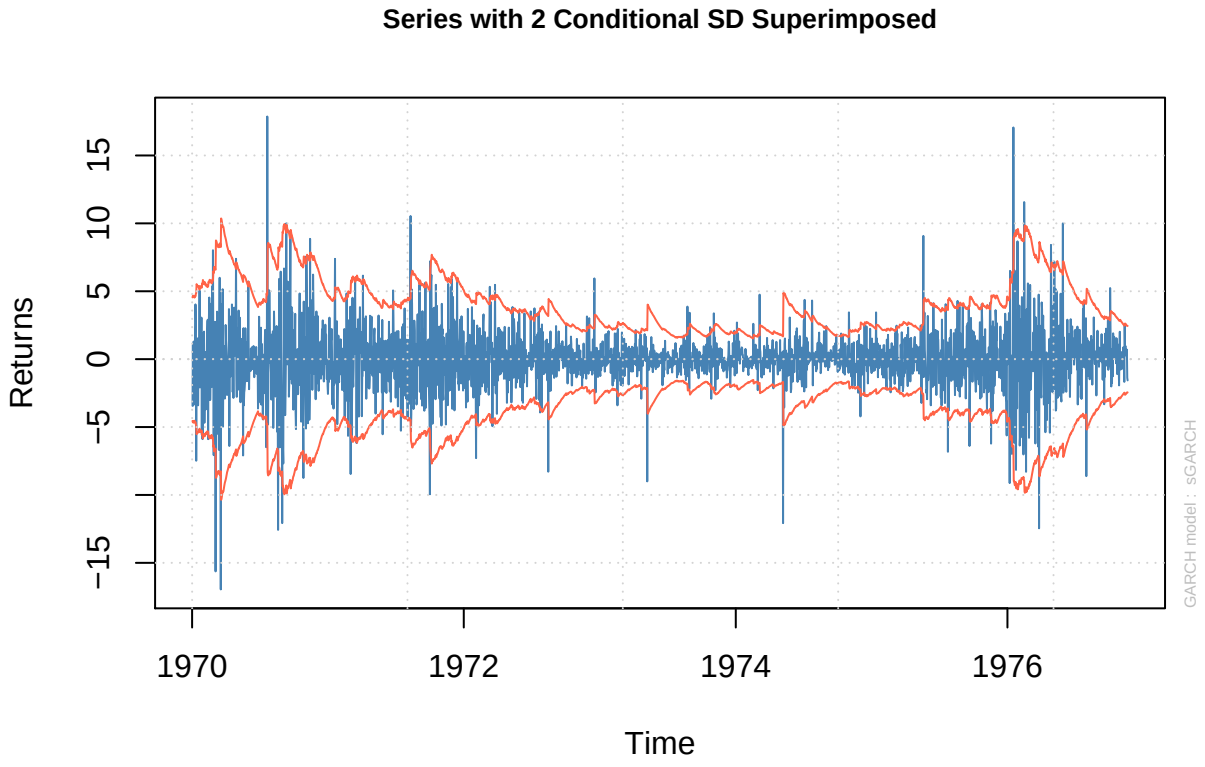
It includes a leverage effect. The eGARCH model is appealing because of the lack of restriction imposed on the parameters. The logarithmic transform complicates the derivation of unbiased forecasts.

The package `rugarch` outputs among other things plenty of test statistics and diagnostics, most of which we won't cover. A very well known one is the value-at-risk (VaR), which measures conditional volatility. The VaR(α) is defined as $\Pr\{L_t>\mathrm{VaR}_t(\alpha)\}=\alpha$, the probability of the loss L_t at time t to exceed the quantile given by $\mathrm{VaR}_t(\alpha)$. For a pure GARCH-type model $Y_t=V_t=\sigma_t\varepsilon_t$, if $\hat{\sigma}_t^{t+1}$ is the one-step ahead conditional standard error, the value-at-risk for time $t+1$ is given by $F^{\leftarrow}(1-\alpha)\hat{\sigma}_t^{t+1}$, where $F^{\leftarrow}(1-\alpha)$ is the $1-\alpha$ quantile of the postulated distribution of the centered residuals ε_t.

One can assess whether the model is adequate by back-testing. Fit the model until time T, forecast one-step ahead variance, verify if the observed volatility is within the confidence band. Then, the number of days for which the loss is greater than the estimated VaR follows a binomial distribution and a likelihood ratio test can be used to check if the number of counts is too high. The function `ugarchroll` provides this. The vignette of the package also discusses a bootstrap method for ARMA-GARCH models.

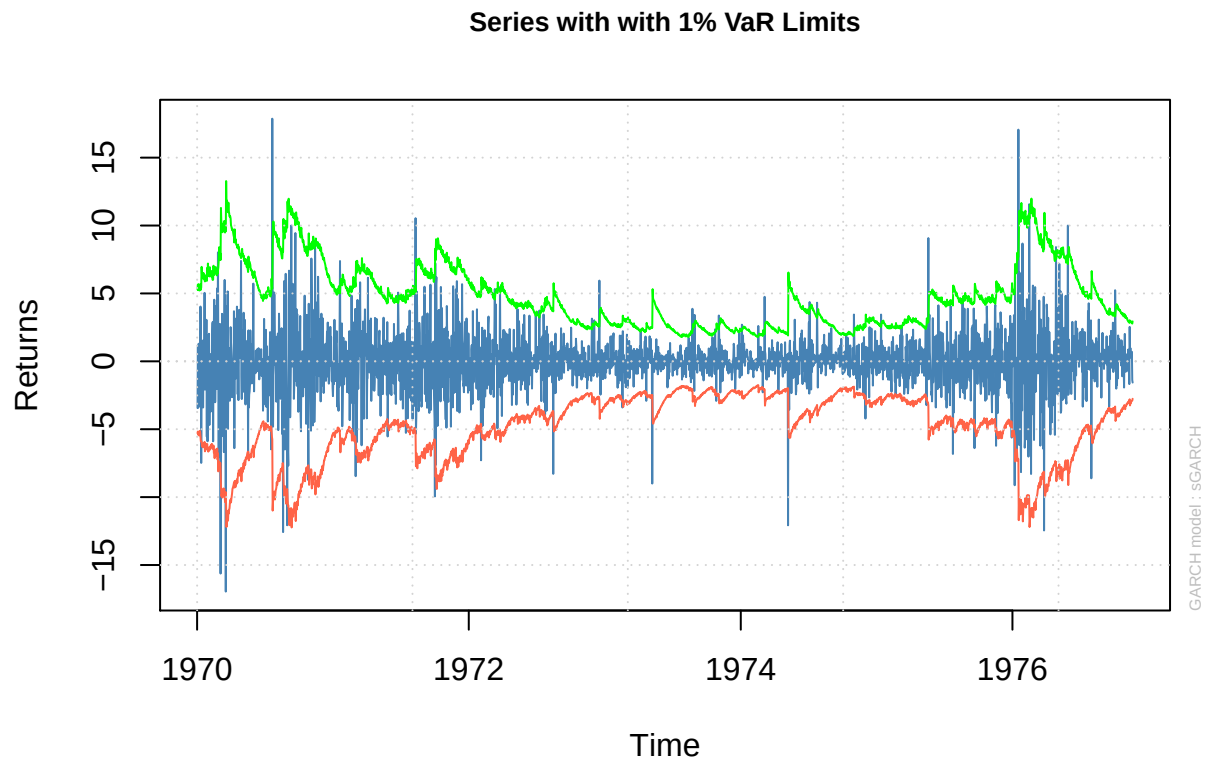
The following plots illustrate pointwise confidence intervals (with the conditional standard deviations $\hat{\sigma}_t$, the 99% VaR, estimates of $\hat{\sigma}_t$ against time, the correlogram of the residuals and squared residuals and a quantile-quantile plot (for the normal distribution, unnormalized). Watch out as the information criterion returned by `rugarch`, AIC and BIC, are scaled by n.

```
# print(model_fit) Plots, can get in interactive mode or use `which` to
# select one plot
plot(model_fit, which = 1)  # series with 95% conf. int (+/- 2 conditional std. dev.)
```

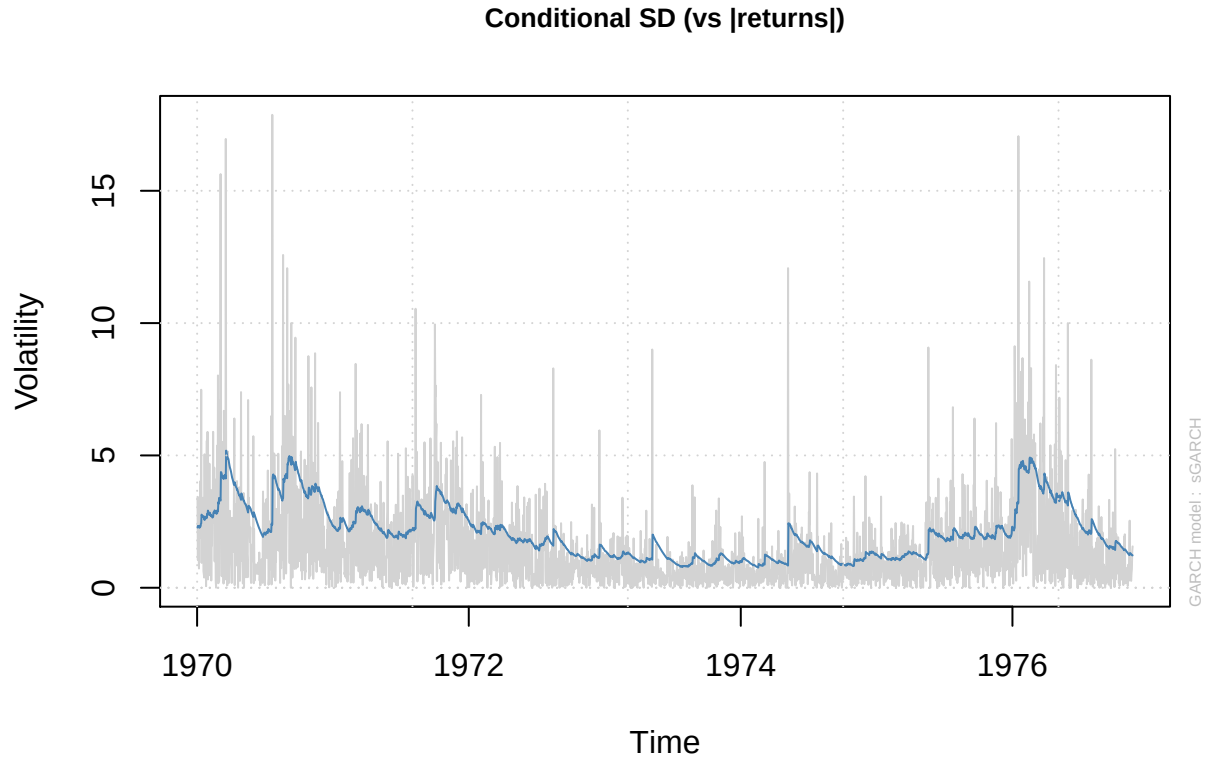



```
plot(model_fit, which = 2)  #VaR 99%
```

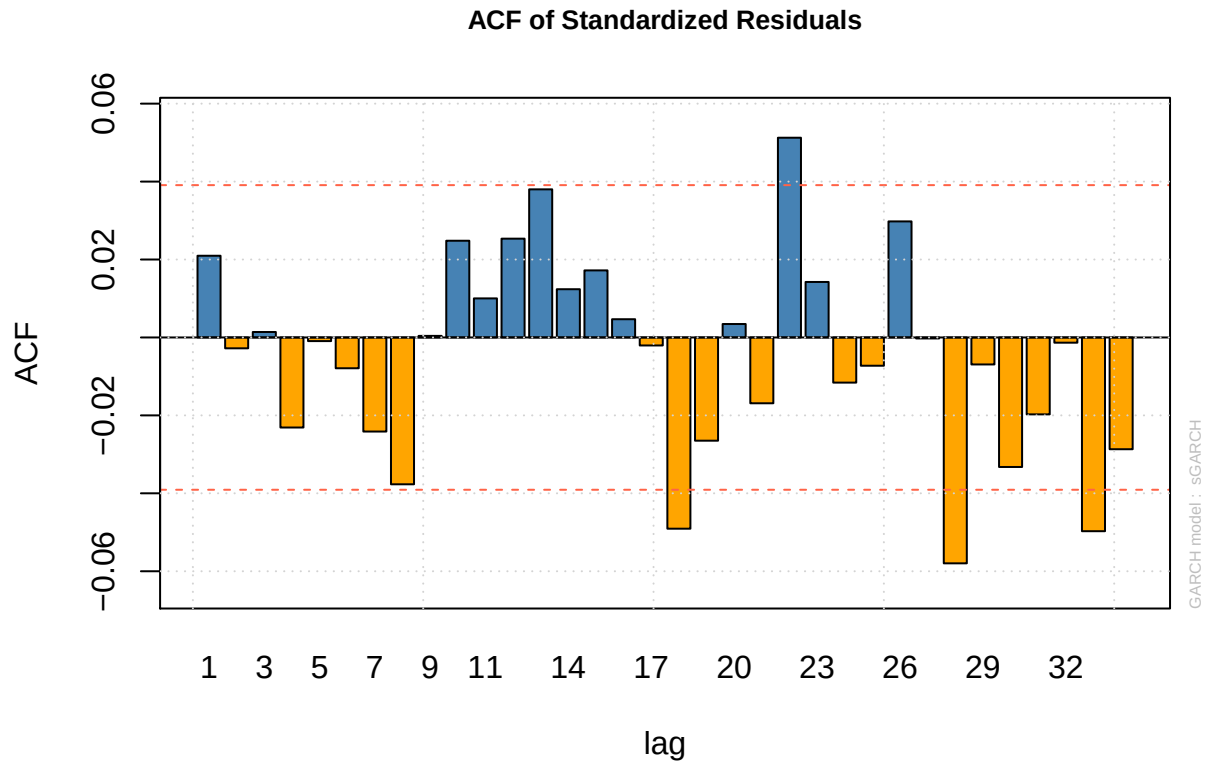
```
please wait...calculating quantiles...
```



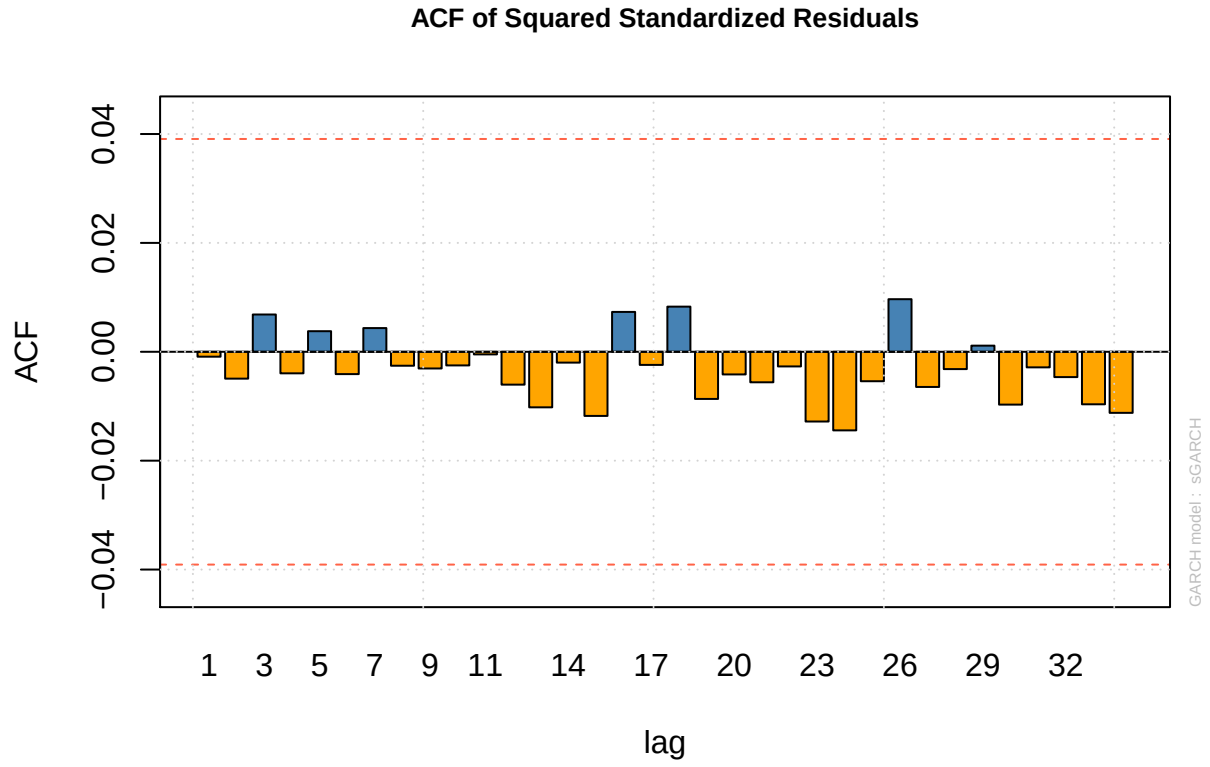

```
plot(model_fit, which = 3)  #Conditional standard derivation
```



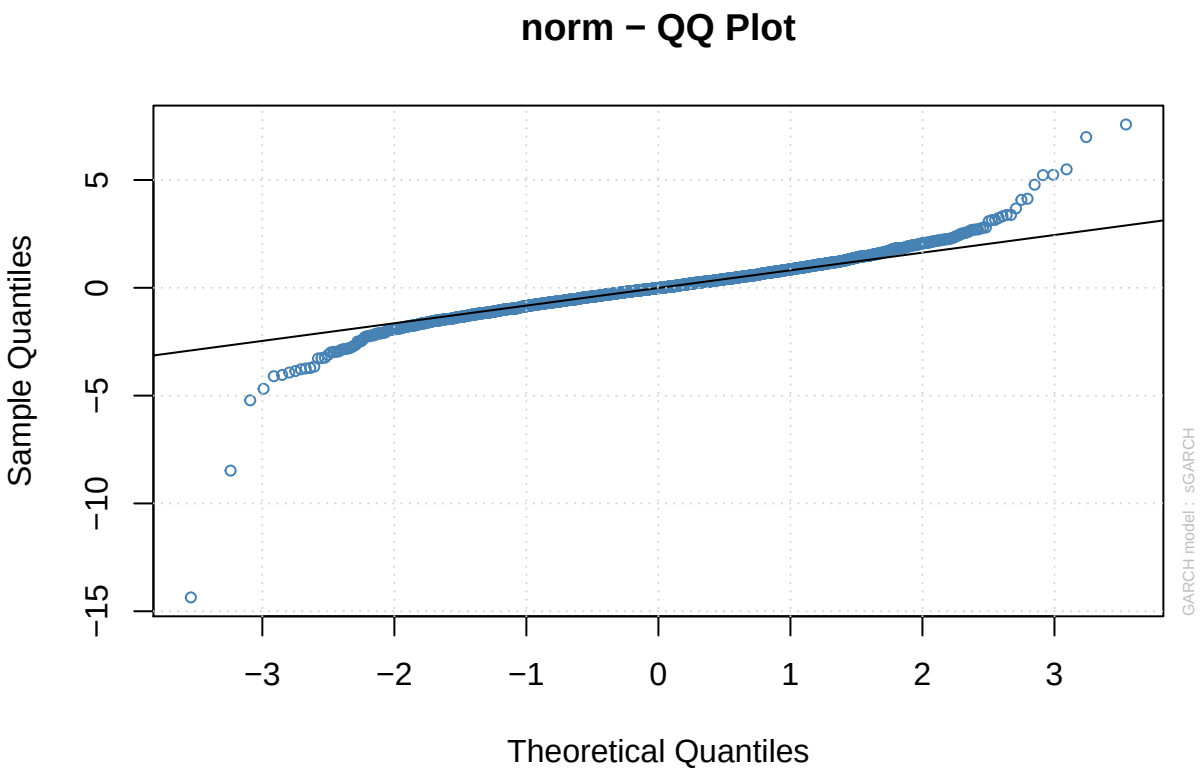

```
plot(model_fit, which = 10)  #ACF of residual
```




```
plot(model_fit, which = 11)  #ACF of squared residuals
```



```
plot(model_fit, which = 9)  #Normal QQ plot
```



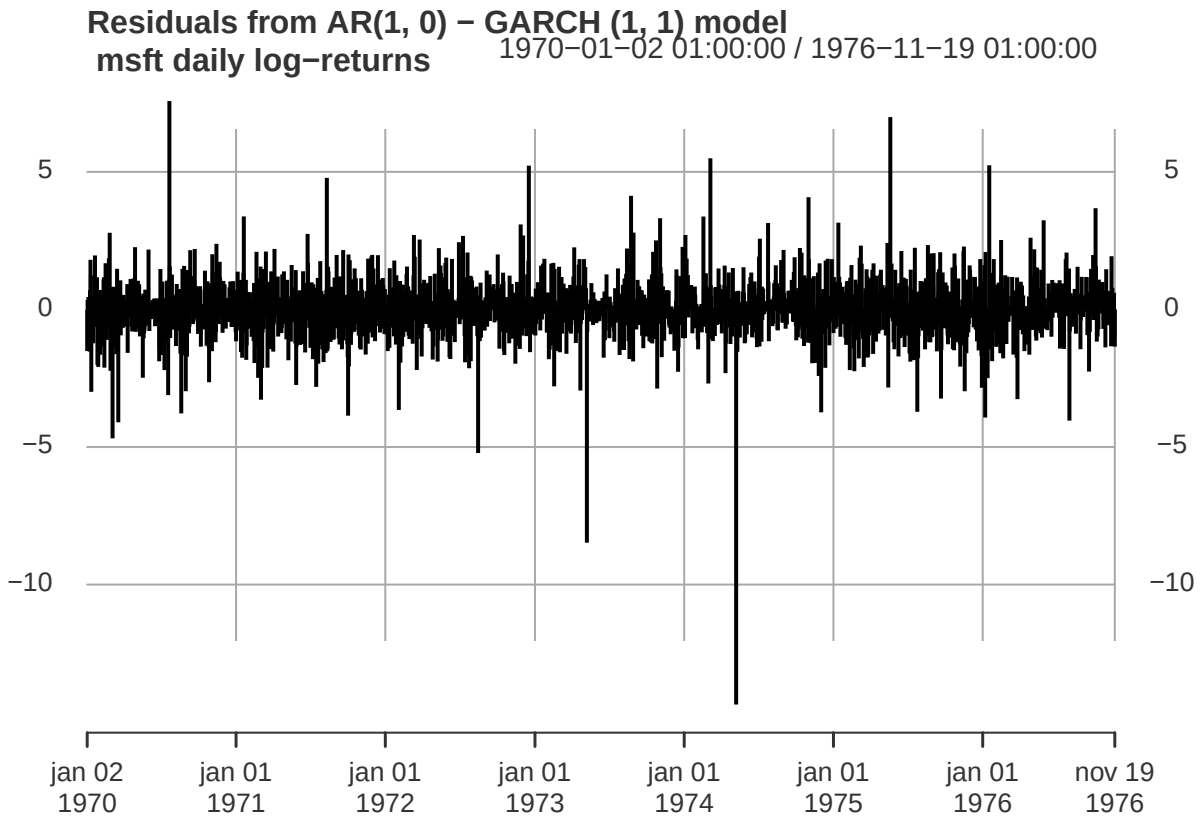

```
# Information criterion
infocriteria(model_fit)
```

```
Akaike       4.174893
Bayes        4.186487
Shibata      4.174885
Hannan-Quinn 4.179101
```

```
model_fit@fit$coef
```

```
          mu          ar1        omega       alpha1        beta1
-0.006676741 -0.072497762  0.003350150  0.036392525  0.963607475
```

```
plot(residuals(model_fit, standardize = TRUE), type = "h", main = "Residuals from AR(1, 0) - GARCH (1, 1
    major.ticks = "years", grid.ticks.on = "years")
```



We note that while the GARCH structure captures the linear dependence well, the normality of the errors is doubtful. One could change the distribution of the errors to Student by changing `distribution.model = "std"`. The fit is then noticeably better, except for some negative returns. Indeed, the GARCH(1,1) model fails to adequately capture the negative returns. There are empirical evidences that the market response to losses is asymmetric, and GARCH fails to capture this. To remediate this, we could consider a variant of the GARCH model, such as eGARCH. Another popular alternative is the GJR-GARCH model, which includes a leverage term. The conditional variance of the GJR-GARCH(1,1) model is of the form

$$\sigma_t^2 = \alpha_0 + (\gamma_1 \text{sign}|\varepsilon_{t-1}| + \alpha_1)\varepsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2).$$

3.3.1 Predictions

It is best to use rolling windows for large time series to perform out-of-sample validation. That is, fit the model to observations $t_k, \ldots, t_{k+N}$, then to $t_{k+1}, \ldots, t_{k+N+1}$ for $k = 1, \ldots, n$. N depends on the context, but for time series it should be roughly 5 years. The function `ugarchroll` can do this for you (but it is computationally intensive, so beware). It is not hard to code your own function (see e.g. this blog post[8]). An bootstrap prediction for returns and volatilities in GARCH models is also implemented; I can provide more details upon request.

The package `fGarch` also has a function `garchFit` and many options, but accessing its output is sometimes more complicated.

```
library(fGarch)
msft_ts <- as.timeSeries(msft.returns)
gjrGARCHfit <- garchFit(~ arma(3, 0) + garch(1, 1), #fit a gjr
  data = msft_ts, trace = FALSE, leverage = TRUE, cond.dist = "QMLE", include.skew = FALSE)
```

[8] http://unstarched.net/2012/12/26/rolling-garch-forecasts/

```
#plot(gjrGARCHfit, which = 3) # series with 95% conf. int (+/- 2 conditional std. dev.)
#plot(gjrGARCHfit, which = 13) #QQ-plot
```

3.3.2 Exercice 3: International Business Machines (IBM) stock

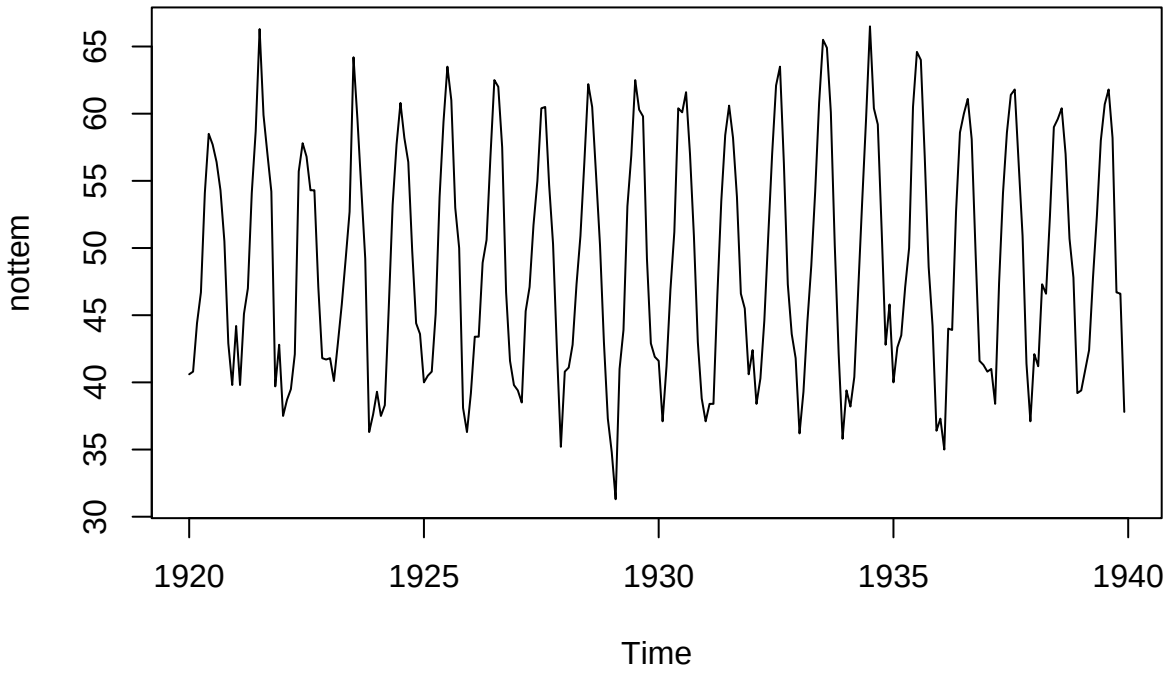
1. Download the daily IBM stocks price from 2003 to 2010 (inclusively). Fit a GARCH(1,1) model with normal errors. Is the model satisfactory? Make sure to check that the GARCH process is not integrated. Does the process display excess kurtosis, relative to that of the normal distribution (for which $\kappa = 3$).
2. If the errors are not normally distributed, experiment with a heavy-tailed distribution. Assess the adequacy of the latter using a QQ-plot.
3. If your returns are asymmetric, try a GARCH-type model that includes a leverage term. Is the fit better?

3.4 Solutions to Exercises

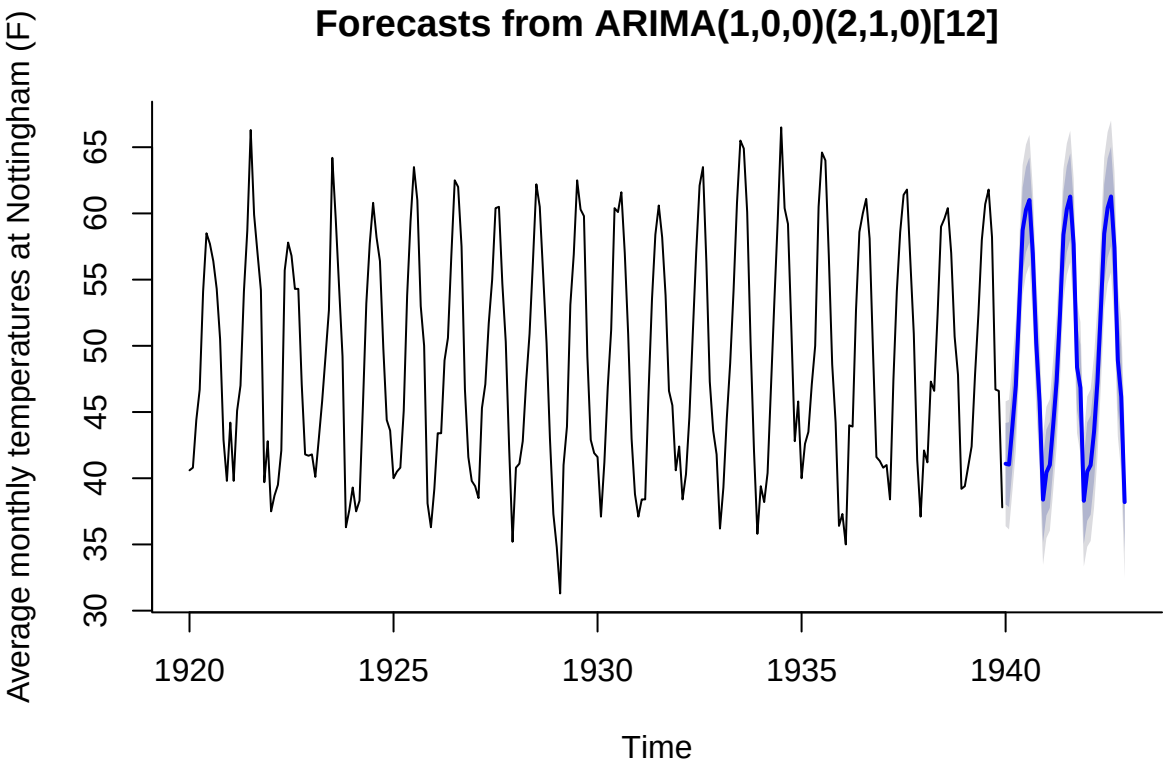
3.4.1 Exercice 1: Nottingham average monthly temperature and Hong Kong monthly exports

1. Fit a SARIMA model on the Nottingham average monthly temperature dataset `nottem` and obtain predictions for the three subsequent years.
2. Fit a SARIMA model to the latter, this time including seasonal dummies as regressors
3. Compare the forecasts from both models
4. Import the dataset `hk_trade_m.csv` and try fitting a suitable SARIMA model to the exports (on the log scale). Contrast the forecasts for different models with their complexity as described by the number of components. What do you notice? Plot the forecasted values on the original scale of the data after back-transforming.

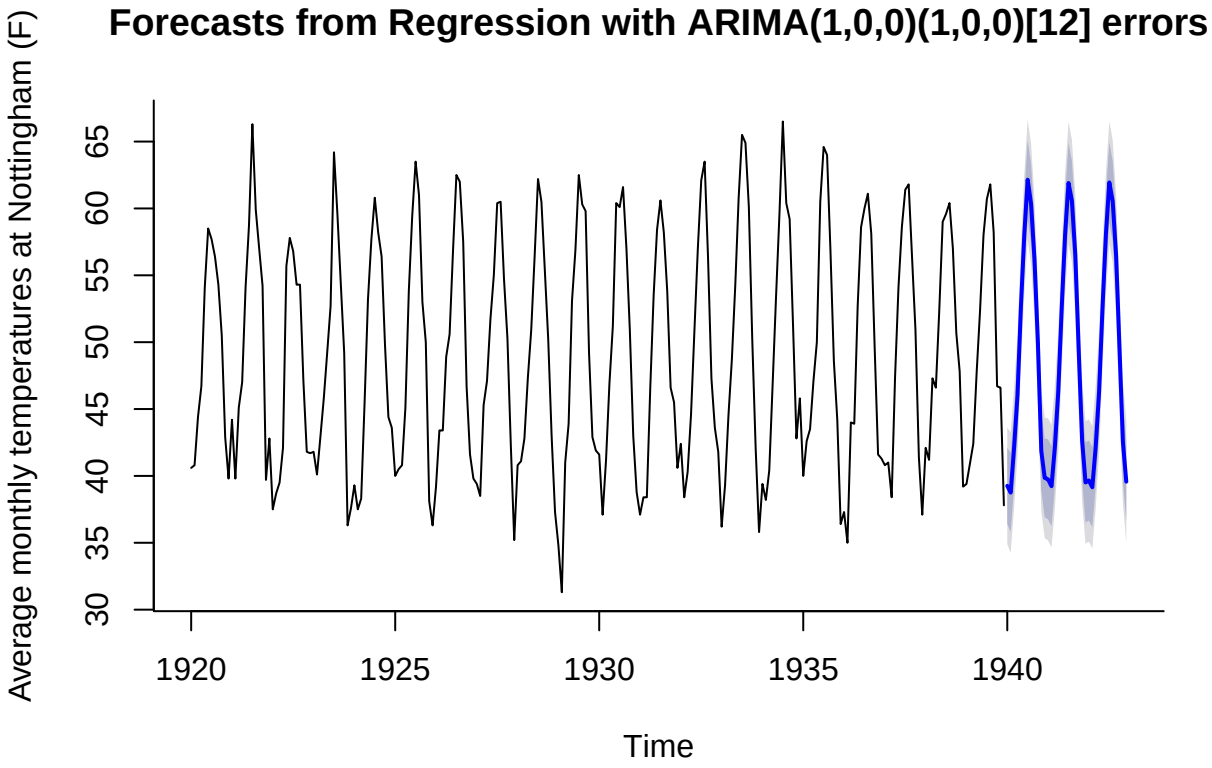
```
# Nottingham monthly air temperature
library(forecast)
plot(nottem)
```



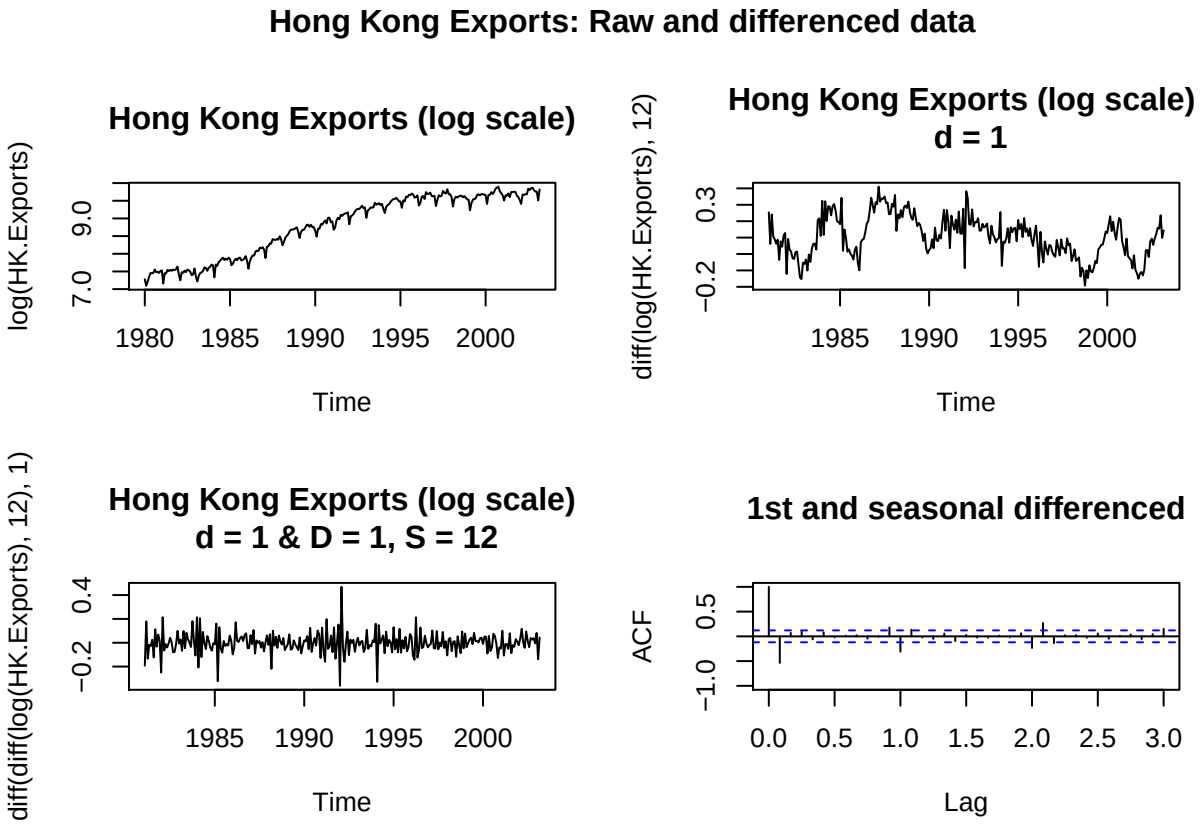
```
not_sar <- Arima(nottem, order = c(1, 0, 0), seasonal = c(2, 1, 0))
not_month <- c((time(nottem) * 12)%%12)
xreg <- sapply(0:11, function(i) {
    not_month == i
})
not_reg <- Arima(nottem, order = c(1, 0, 0), seasonal = c(1, 0, 0), xreg = xreg,
    include.mean = FALSE)  #remove mean, otherwise collinear

plot(forecast(not_sar, h = 36), ylab = "Average monthly temperatures at Nottingham (F)",
    bty = "l", xlab = "Time")
```



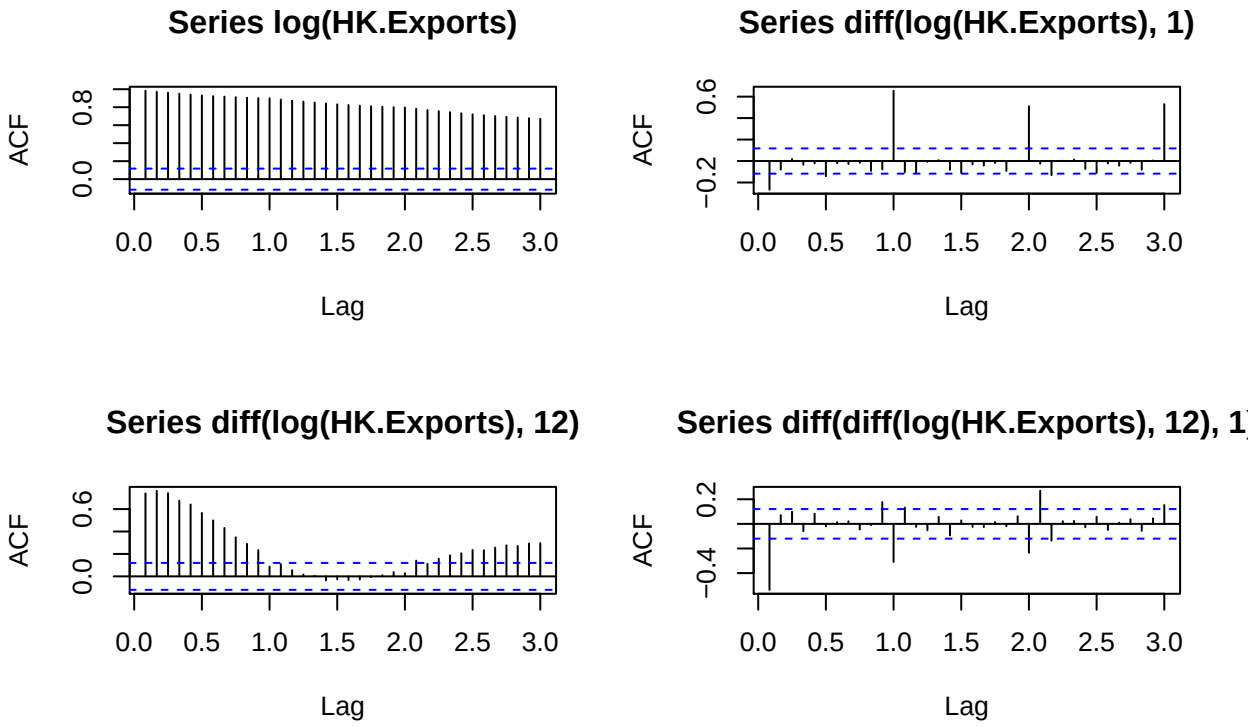

```
plot(forecast(not_reg, h = 36, xreg = xreg[1:36, ]), ylab = "Average monthly temperatures at Nottingham
    bty = "l", xlab = "Time")
```




```r
# Hong Kong trade - code and analysis from Prof. David A. Stephens
HKTrade <- read.csv("http://sma.epfl.ch/~lbelzile/math342/hk_trade_m.csv", header = FALSE)
library(forecast)
HK.Exports <- ts(HKTrade[, 2], start = c(1980, 1), freq = 12)
HK.Imports <- ts(HKTrade[, 3], start = c(1980, 1), freq = 12)

par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))
plot(log(HK.Exports), main = "Hong Kong Exports (log scale)")
plot(diff(log(HK.Exports), 12), main = "Hong Kong Exports (log scale) \n d = 1")
plot(diff(diff(log(HK.Exports), 12), 1), main = "Hong Kong Exports (log scale)\n d = 1 & D = 1, S = 12"
acf(diff(diff(log(HK.Exports), 12), 1), lag.max = 36, ylim = range(-1, 1), main = "1st and seasonal dif
title("Hong Kong Exports: Raw and differenced data", outer = TRUE)
```



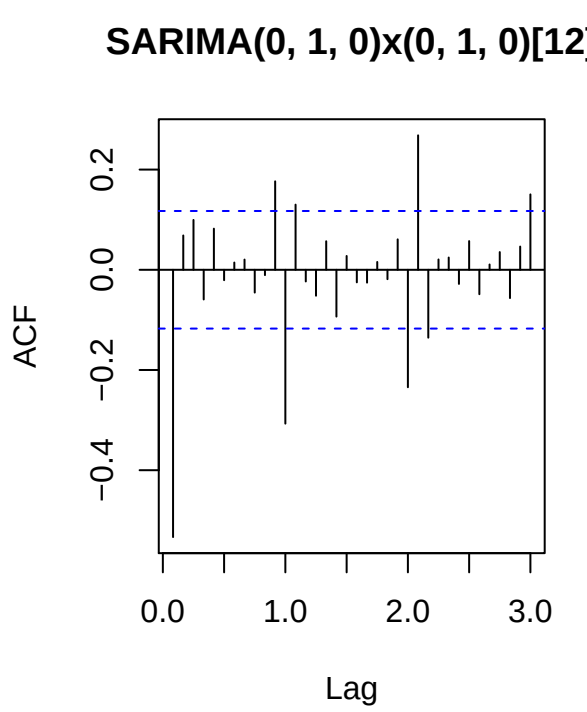

```
# Differencing for the log-series
TSA::acf(log(HK.Exports), lag.max = 36)
TSA::acf(diff(log(HK.Exports), 1), lag.max = 36)
TSA::acf(diff(log(HK.Exports), 12), lag.max = 36)
TSA::acf(diff(diff(log(HK.Exports), 12), 1), lag.max = 36)
```

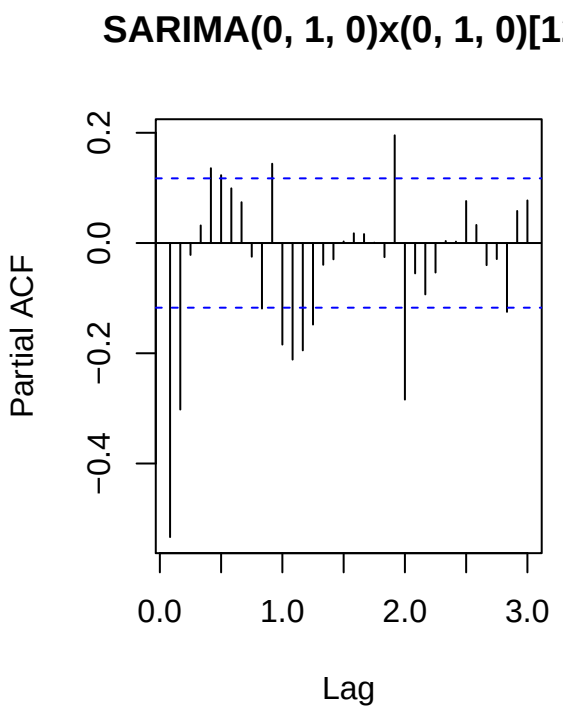



```
par(mfrow = c(1, 2), oma = c(0, 0, 2, 0))
Yt <- log(HK.Exports)
# Bottom-up modelling with SARIMA models
fit.HK.1 <- Arima(Yt, c(0, 1, 0), seasonal = list(order = c(0, 1, 0), period = 12),
    method = "ML")
summary(fit.HK.1)
```

```
Series: Yt
ARIMA(0,1,0)(0,1,0)[12]

sigma^2 estimated as 0.0081:  log likelihood=263.12
AIC=-524.24   AICc=-524.23   BIC=-520.66

Training set error measures:
                      ME       RMSE        MAE           MPE      MAPE
Training set -0.0004542289 0.08787689 0.06156272 -0.008057074 0.7100876
                  MASE       ACF1
Training set 0.458178 -0.5335023
```

```
TSA::acf(residuals(fit.HK.1), lag.max = 36, main = "SARIMA(0, 1, 0)x(0, 1, 0)[12]")
pacf(residuals(fit.HK.1), lag.max = 36, main = "SARIMA(0, 1, 0)x(0, 1, 0)[12]")
```

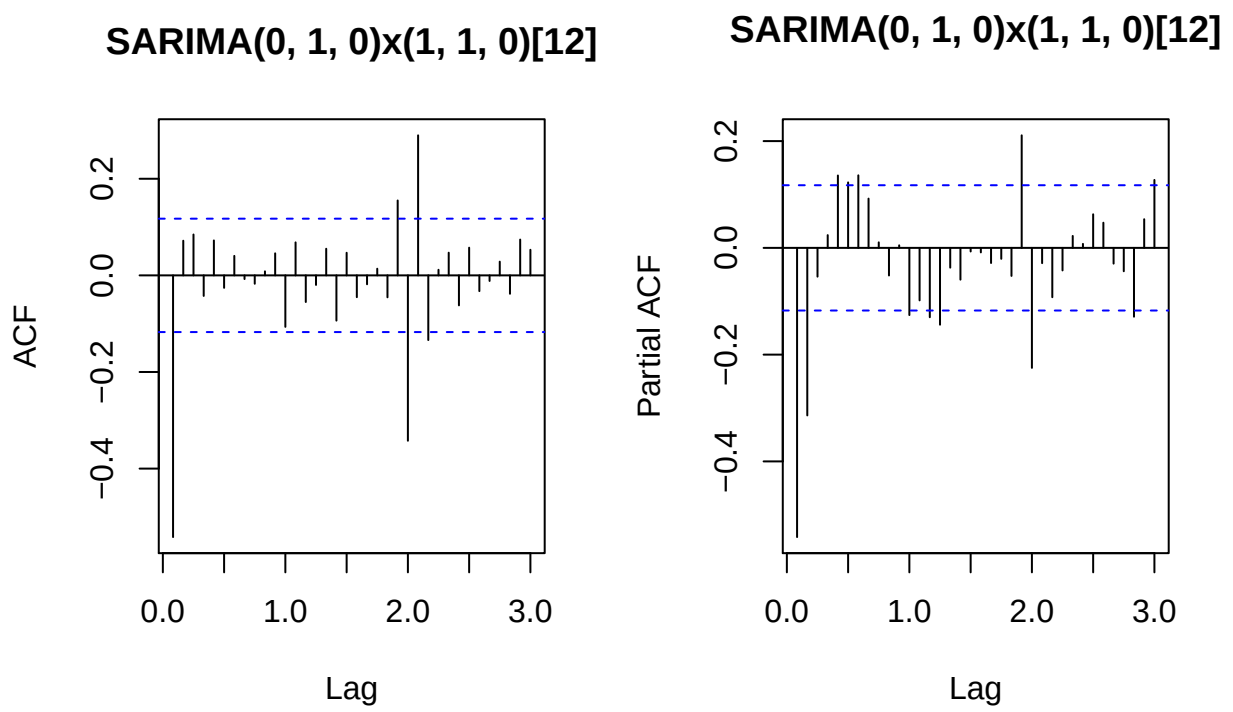





```r
# Remove the remaining seasonality
fit.HK.2 <- Arima(Yt, c(0, 1, 0), seasonal = list(order = c(1, 1, 0), period = 12),
    method = "ML")
summary(fit.HK.2)
```

```
Series: Yt
ARIMA(0,1,0)(1,1,0)[12]

Coefficients:
         sar1
      -0.3300
s.e.   0.0595

sigma^2 estimated as 0.007259:  log likelihood=277.5
AIC=-551   AICc=-550.96   BIC=-543.84

Training set error measures:
                       ME       RMSE        MAE         MPE      MAPE
Training set -0.0006534918 0.08303735 0.05940653 -0.01068992 0.6839377
                   MASE       ACF1
Training set 0.4421306 -0.5417701
```

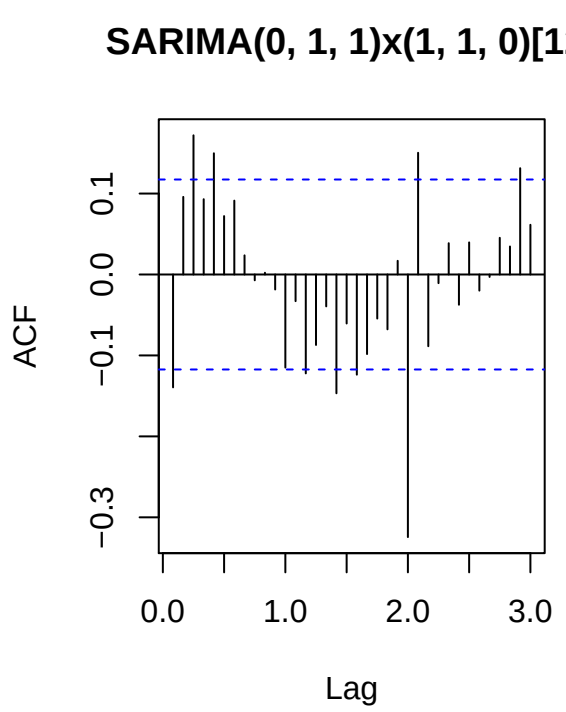
```r
TSA::acf(residuals(fit.HK.2), lag.max = 36, main = "SARIMA(0, 1, 0)x(1, 1, 0)[12]")
pacf(residuals(fit.HK.2), lag.max = 36, main = "SARIMA(0, 1, 0)x(1, 1, 0)[12]")
```

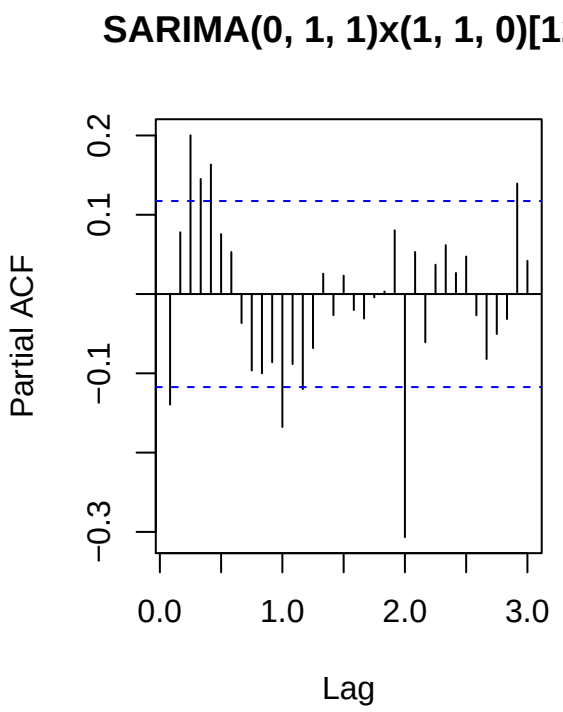



```
# Remove moving average component at lag 1
fit.HK.3 <- Arima(Yt, c(0, 1, 1), seasonal = list(order = c(1, 1, 0), period = 12),
    method = "ML")
summary(fit.HK.3)
```

```
Series: Yt
ARIMA(0,1,1)(1,1,0)[12]

Coefficients:
          ma1     sar1
      -0.5726  -0.3698
s.e.   0.0385   0.0585

sigma^2 estimated as 0.004857:  log likelihood=331.1
AIC=-656.2   AICc=-656.11   BIC=-645.45

Training set error measures:
                      ME       RMSE       MAE         MPE      MAPE
Training set -0.001208245 0.06778969 0.0518109 -0.01640749 0.5963856
                   MASE       ACF1
Training set 0.3856004 -0.1394624
```

```
TSA::acf(residuals(fit.HK.3), lag.max = 36, main = "SARIMA(0, 1, 1)x(1, 1, 0)[12]")
pacf(residuals(fit.HK.3), lag.max = 36, main = "SARIMA(0, 1, 1)x(1, 1, 0)[12]")
```

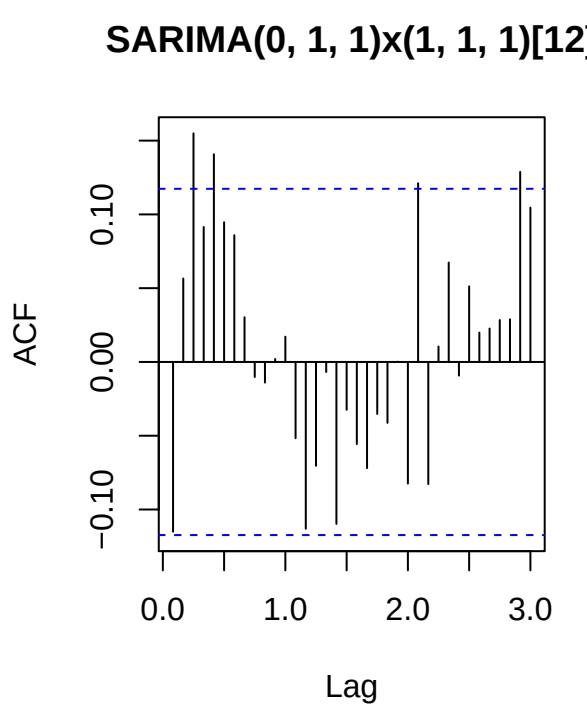


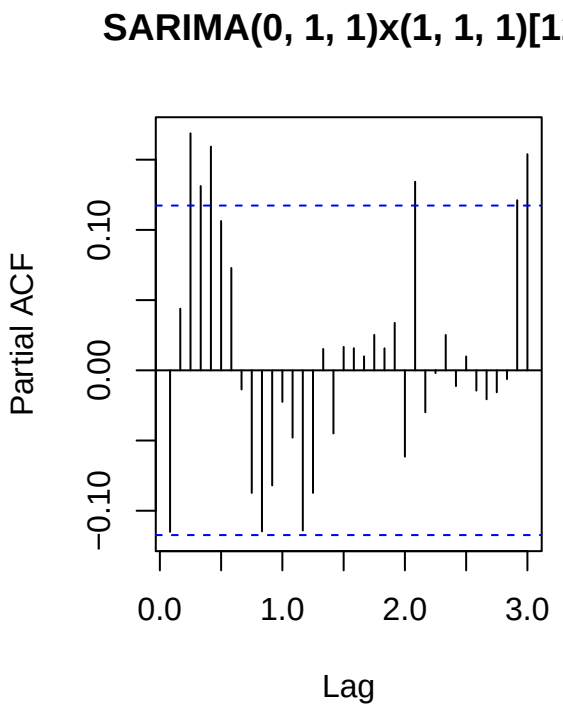



```r
# Remove seasonal component at lag 24
fit.HK.4 <- Arima(Yt, c(0, 1, 1), seasonal = list(order = c(1, 1, 1), period = 12),
    method = "ML")
summary(fit.HK.4)
```

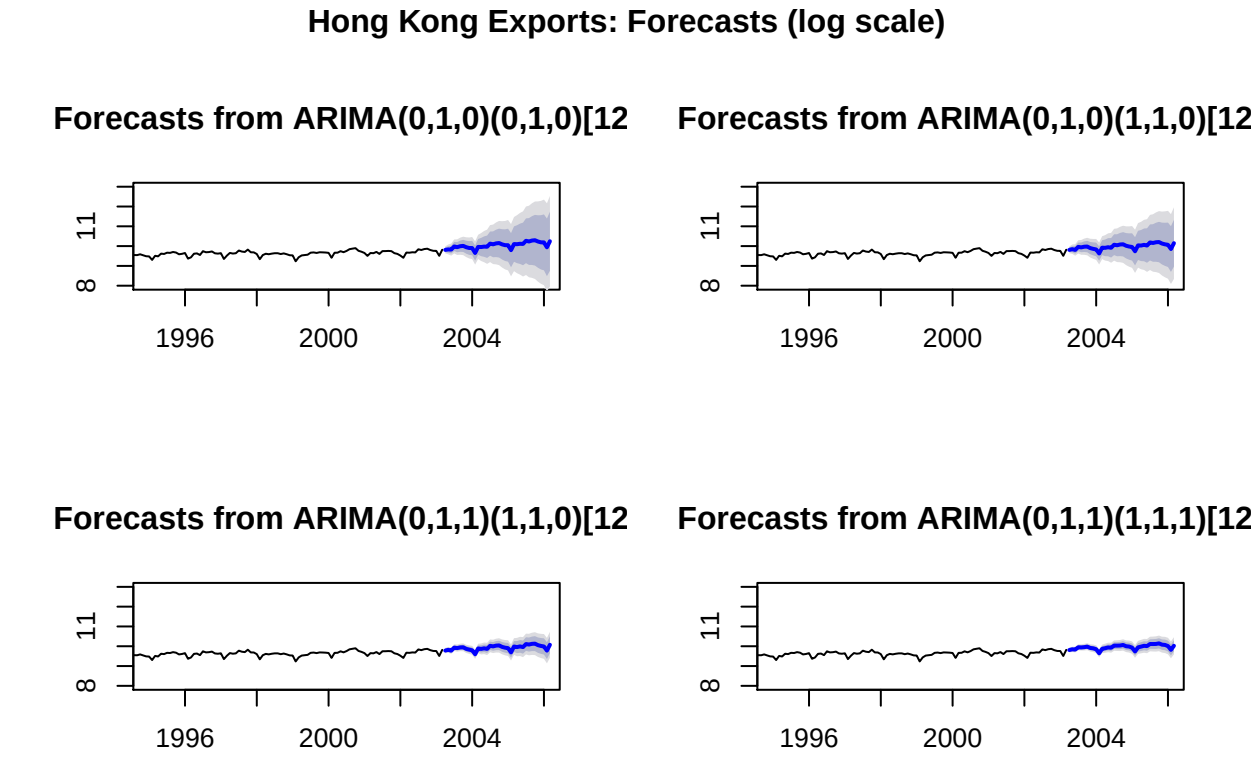
```
Series: Yt
ARIMA(0,1,1)(1,1,1)[12]

Coefficients:
          ma1     sar1     sma1
      -0.5697  0.2034  -0.8641
s.e.   0.0392  0.0845   0.0621

sigma^2 estimated as 0.003824:  log likelihood=358.14
AIC=-708.28   AICc=-708.13   BIC=-693.95

Training set error measures:
                      ME       RMSE        MAE         MPE      MAPE
Training set -0.002437551 0.06003768 0.04482937 -0.02675703 0.5149574
                   MASE       ACF1
Training set 0.3336407 -0.1150611
```

```r
TSA::acf(residuals(fit.HK.4), lag.max = 36, main = "SARIMA(0, 1, 1)x(1, 1, 1)[12]")
pacf(residuals(fit.HK.4), lag.max = 36, main = "SARIMA(0, 1, 1)x(1, 1, 1)[12]")
```

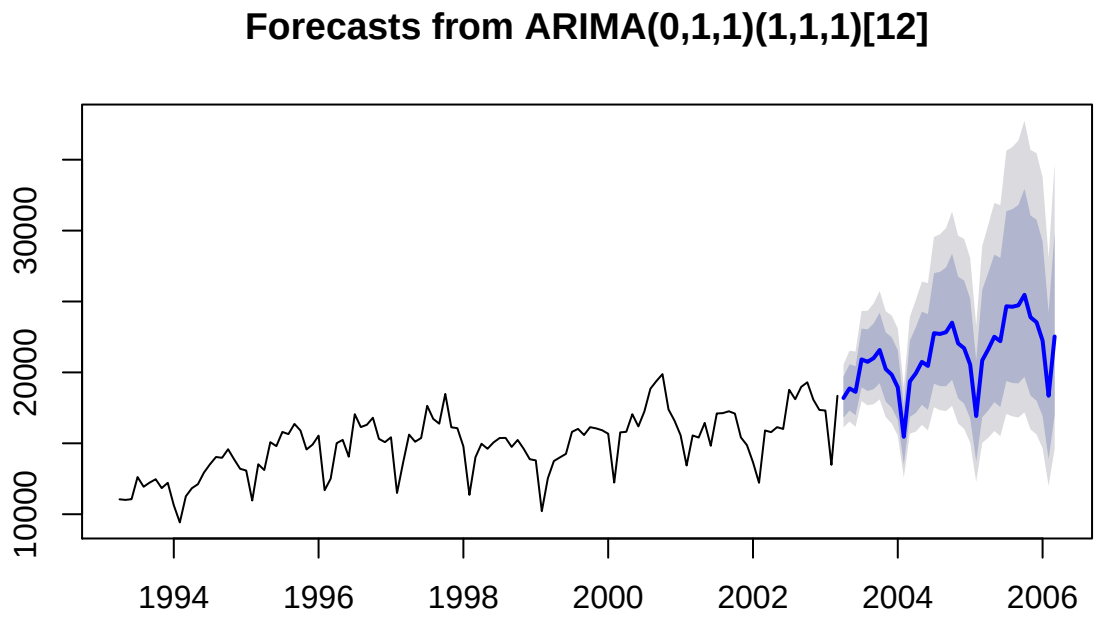





```r
# Limits of the SARIMA model - the trend was not linear there are residual
# features despite differencing little can be done at this stage
par(mfrow = c(2, 2))
plot(forecast(fit.HK.1, h = 36), ylim = range(8, 13), xlim = range(1995:2006))
plot(forecast(fit.HK.2, h = 36), ylim = range(8, 13), xlim = range(1995:2006))
plot(forecast(fit.HK.3, h = 36), ylim = range(8, 13), xlim = range(1995:2006))
plot(forecast(fit.HK.4, h = 36), ylim = range(8, 13), xlim = range(1995:2006))
title("Hong Kong Exports: Forecasts (log scale)", outer = TRUE)
```




```
# Forecasts on the original scale
par(mfrow = c(1, 1))
# lambda <- BoxCox.lambda(HK.Exports, method = 'loglik')
fit.HK.4b <- Arima(HK.Exports, c(0, 1, 1), lambda = 0, seasonal = list(order = c(1,
    1, 1), period = 12), method = "ML")
plot(forecast(fit.HK.4b, h = 36), include = 120)
```



3.4.2 Exercice 2: Lake Erie and Lake Huron levels

1. Fit a linear model to the January observations of the Lake Erie data of the form $y_t = \beta_0 + \beta_1 t + \varepsilon_t$, ignoring the temporal dependence. Test the null hypothesis that the trend is not significant.
2. Use a parametric sieve bootstrap with normal errors to assess the presence of a trend in the Lake Huron dataset. Report your conclusion as a P-value.
3. Recall the estimation of the Lake Huron level in Practical 2. There, we saw that fitting an ARMA(2,1) led to a parameter value of $\theta_1 \approx 1$. Using a parametric bootstrap, test the hypothesis that the parameter $\theta_1 = 0$.

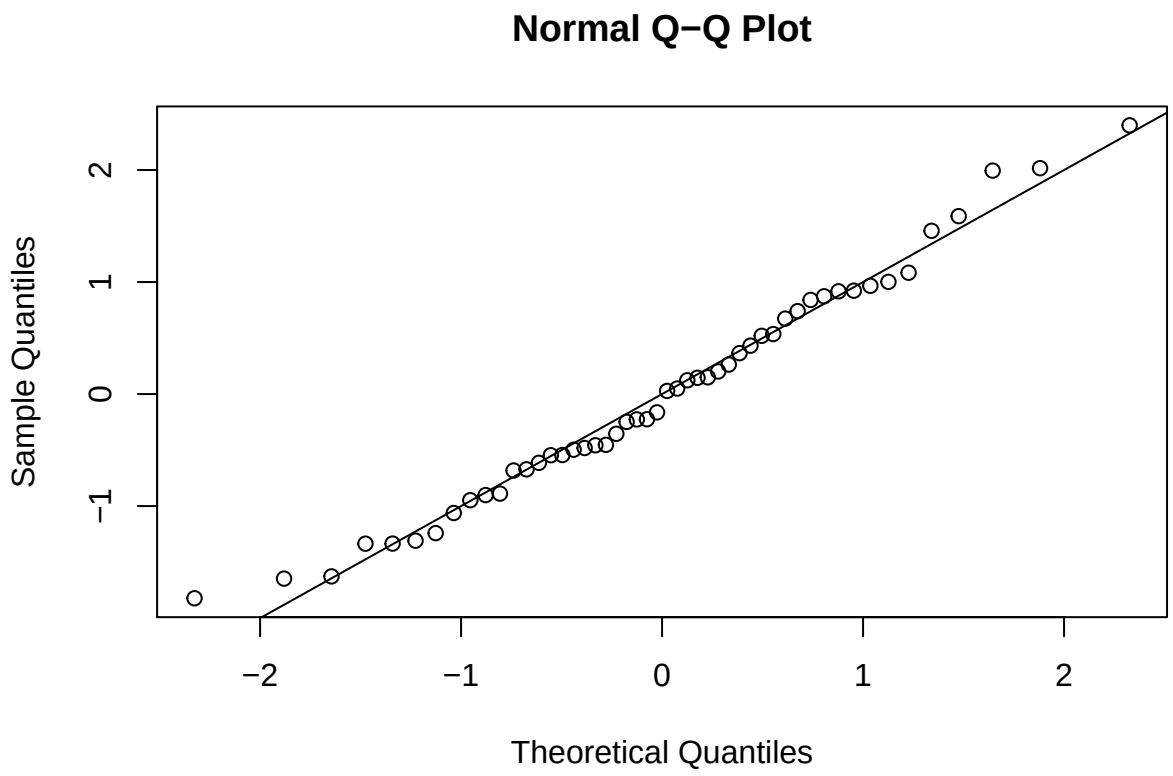
```
# From practical series 2
library(forecast)
lake <- read.csv("http://sma.epfl.ch/~lbelzile/math342/LakeErie.csv", header = TRUE,
    stringsAsFactors = FALSE, sep = ";")
lake.ts <- subset(ts(lake[, 2], start = c(1920, 1), frequency = 12), month = 1)
lmfit <- lm(lake.ts ~ scale(time(lake.ts)))
summary(lmfit)$coefficients[2, 3]
```

```
[1] 2.150708
```

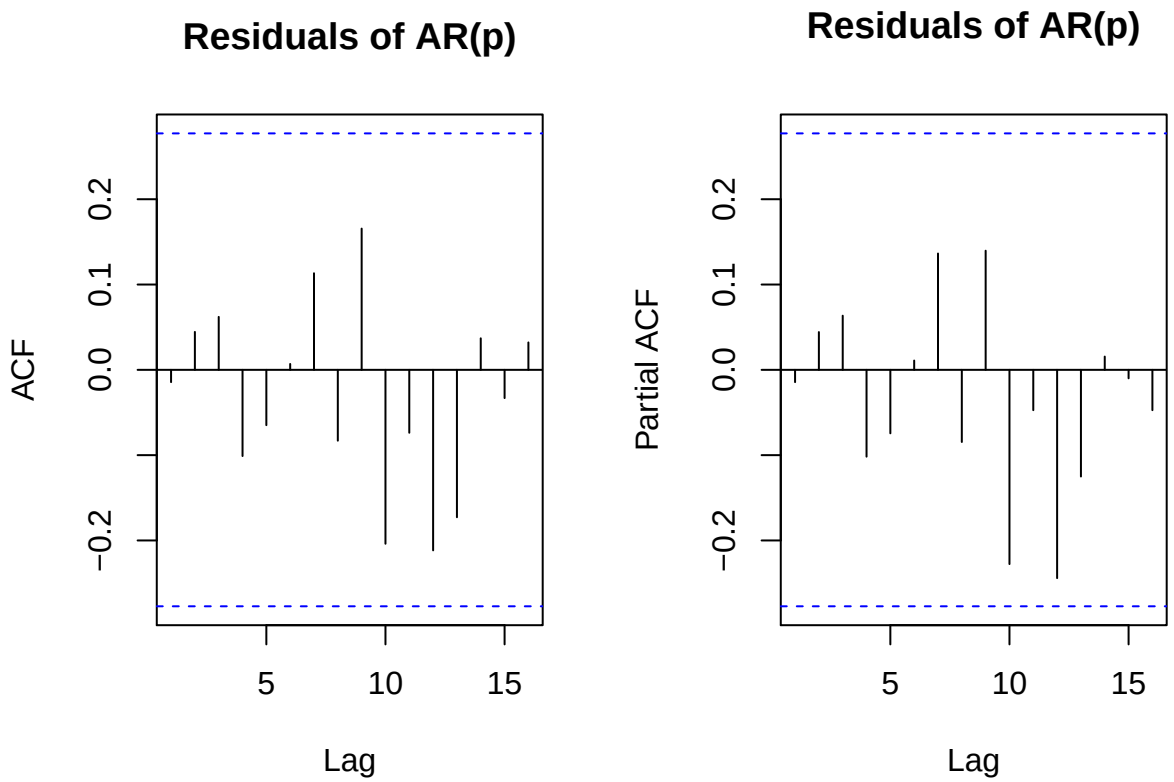
```
# Test statistic, with one sided p-value
2 * pt(summary(lmfit)$coefficients[2, 1]/summary(lmfit)$coefficients[2, 2],
    df = lmfit$df.residual, lower.tail = FALSE)
```

```
[1] 0.03656358
```

```
p <- 5
fit_lake <- Arima(lake.ts, order = c(p, 0, 0), include.mean = TRUE, include.drift = TRUE)

boot_stat <- function(bts) {
    ts_ar <- Arima(bts, order = c(p, 0, 0), include.mean = TRUE, include.drift = TRUE)
    ts_ar$coef["drift"]
}

# Check normality assumption and residual dependence
qqnorm(scale(resid(fit_lake)))
abline(a = 0, b = 1)
```



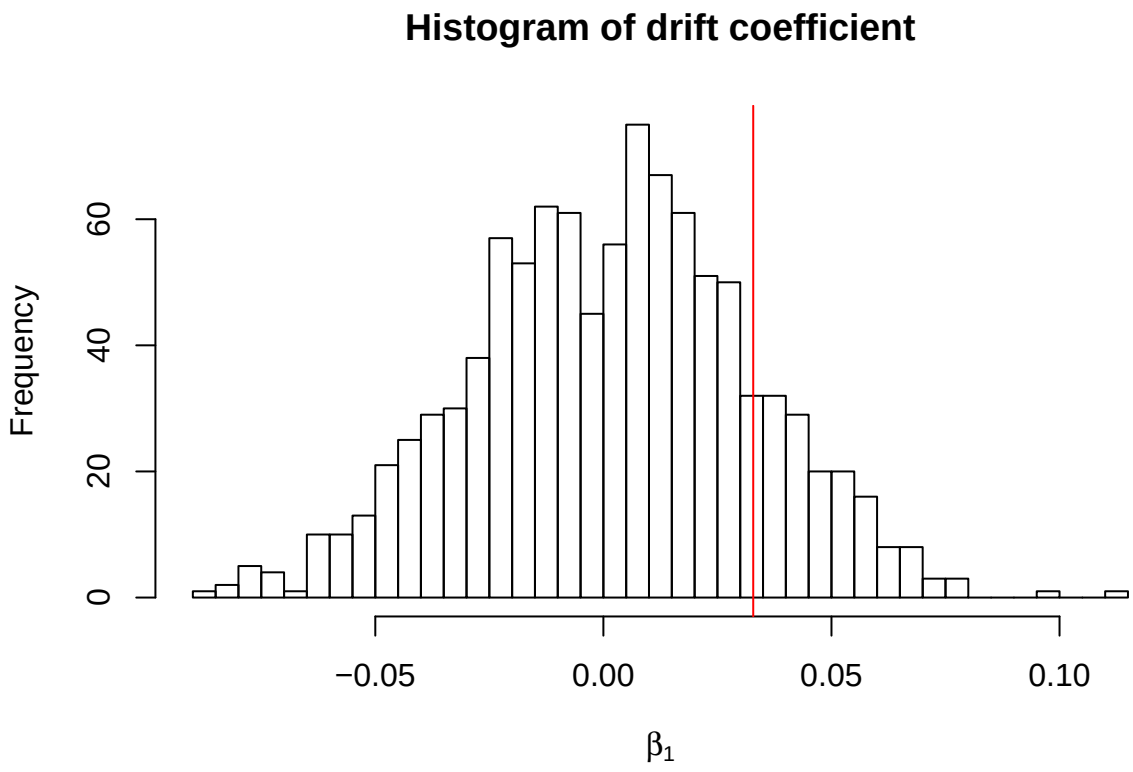

```
par(mfrow = c(1, 2))
TSA::acf(resid(fit_lake), main = "Residuals of AR(p)")
pacf(resid(fit_lake), main = "Residuals of AR(p)")
```



```
par(mfrow = c(1, 1))

# Coefficient estimated using an AR model

# Under H0, the drift is zero
ts_null <- Arima(lake.ts, order = c(p, 0, 0), include.mean = TRUE, include.drift = FALSE)
B <- 999
boot_res <- c(boot_stat(lake.ts), replicate(n = B, expr = Arima(arima.sim(model = list(ar = ts_null$mod
    order = c(p, 0, 0)), n = length(lake.ts), sd = sqrt(fit_lake$sigma2)) +
    ts_null$coef["intercept"], order = c(p, 0, 0), include.mean = TRUE, include.drift = TRUE)$coef["dri
# P-value
1 - rank(boot_res)[1]/(B + 1)
```

```
drift
0.156
```

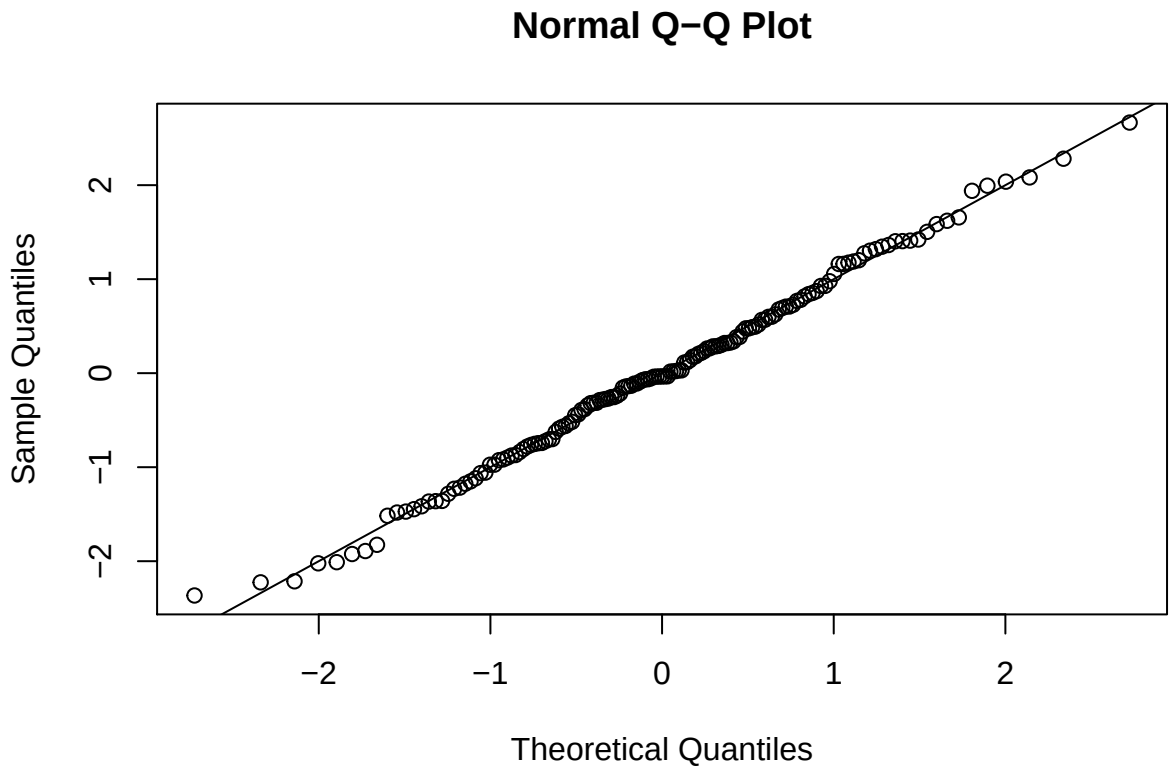
```
# Not statistically significant at level 5%, barely at 10%
hist(boot_res, breaks = 50, main = "Histogram of drift coefficient", xlab = expression(beta[1]))
abline(v = boot_res[1], col = 2)
```



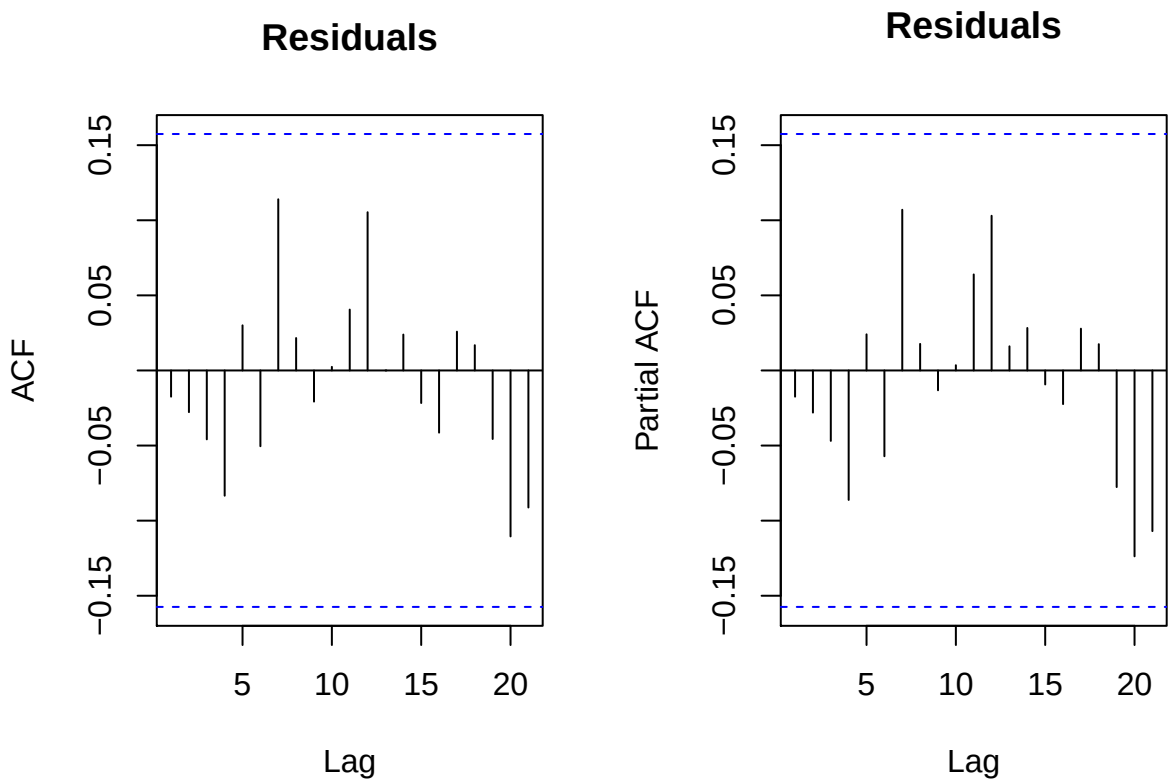
The test statistic for the Wald test is $(\hat{\beta} - \hat{\beta}_0)/\widehat{\text{se}}(\hat{\beta})$, which is 2.150708. The latter follows a t distribution with $n-k$ degrees of freedom under the null. If there is temporal dependence, we have less new units of information, since the data is autocorrelated.

The choice of the order for the AR(p) approximation in the AR-sieve bootstrap can be delicate, but AIC typically leads to adequate choices. Here, taking $p = 2$ seems sufficient.

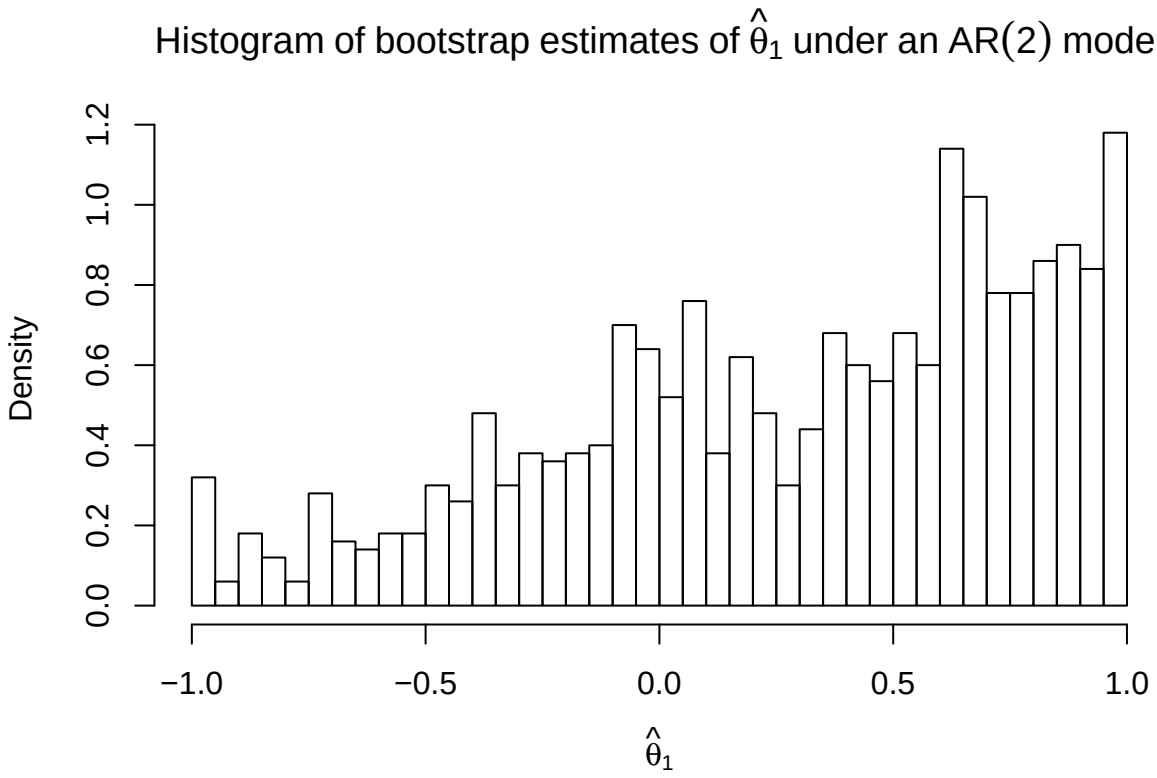
```r
# Licensed under the Creative Commons attribution-noncommercial license
# Adapted from work of Prof. Edward Ionides,
# https://ionides.github.io/531w16/notes05/notes5.html

library(lubridate)
library(forecast)
# Data from
# http://www.glerl.noaa.gov/data/dashboard/data/levels/mGauge/miHuronMog.csv
dat <- read.csv("http://sma.epfl.ch/~lbelzile/math342/huron_depth.csv", sep = ",",
    header = TRUE, skip = 2)
dat$Date <- strptime(dat$Date, "%m/%d/%Y")
dat$year <- with(dat, year(Date))
dat$month <- with(dat, month(Date))  #as.numeric(format(dat$Date, format = '%m'))

## Subsample values to take only January data
dat <- subset(dat, month == 1)
huron_depth <- dat$Average
year <- dat$year
huron_arma21 <- Arima(huron_depth, order = c(2, 0, 1))
huron_arma20 <- Arima(huron_depth, order = c(2, 0, 0))
qqnorm(scale(resid(huron_arma20)))
abline(a = 0, b = 1)
```




```r
par(mfrow = c(1, 2))
TSA::acf(resid(huron_arma20), main = "Residuals")
pacf(resid(huron_arma20), main = "Residuals")
```




```
par(mfrow = c(1, 1))
set.seed(57892330)
B <- 1000
params <- coef(huron_arma20)
ar <- params[grep("^ar", names(params))]
ma <- params[grep("^ma", names(params))]
intercept <- params["intercept"]
sigma <- sqrt(huron_arma21$sigma2)
theta <- matrix(NA, nrow = B, ncol = length(coef(huron_arma21)), dimnames = list(NULL,
    names(coef(huron_arma21))))
theta[1, ] <- huron_arma21$coef
for (b in 2:B) {
    Y_b <- arima.sim(list(ar = ar, ma = ma), n = length(huron_depth), sd = sigma) +
        intercept
    theta[b, ] <- coef(Arima(Y_b, order = c(2, 0, 1), method = "ML", include.mean = TRUE,
        include.drift = FALSE))
}
hist(theta[, "ma1"], freq = FALSE, breaks = 30, main = expression(Histogram ~
    of ~ bootstrap ~ estimates ~ of ~ hat(theta)[1] ~ under ~ an ~ AR(2) ~ model),
    xlab = expression(hat(theta)[1]))
```




```
# P-value - here the result is a bit spurious because of potential ties
# since the distribution is discrete and there are ties
1 - rank(abs(theta[, 3]))[1]/B
```

```
[1] 0.025
```

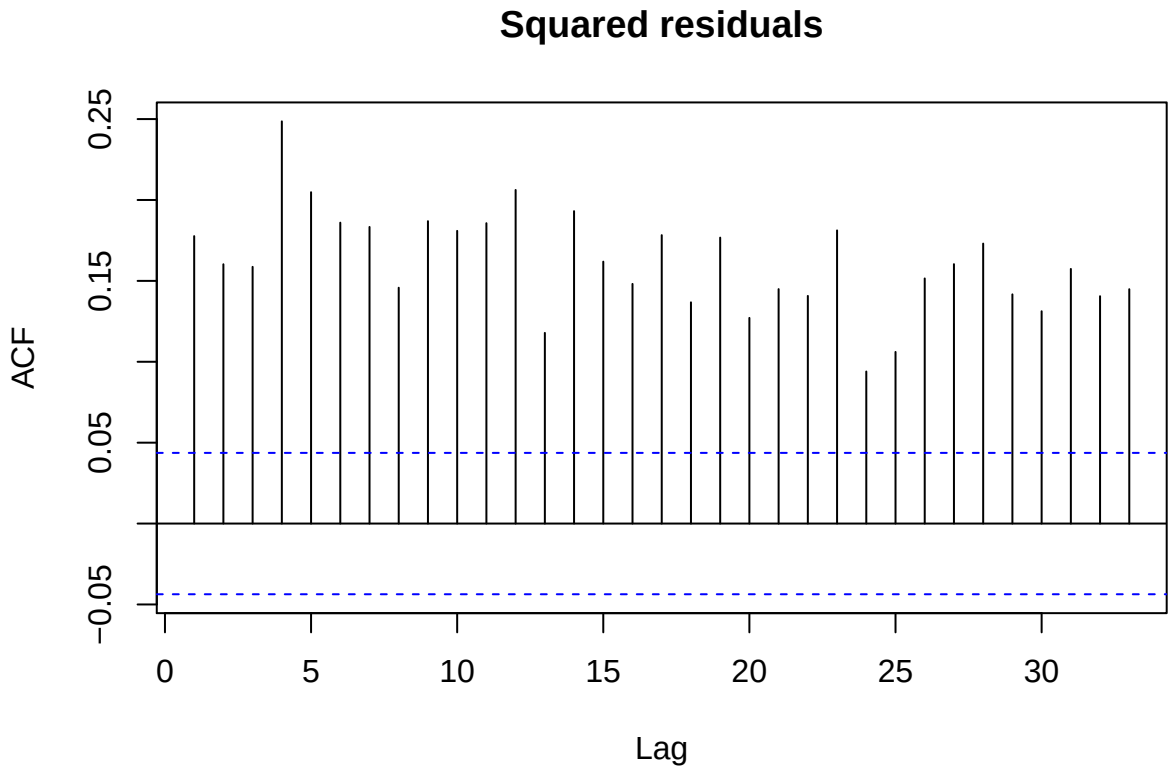
3.4.3 Exercice 3: International Business Machines (IBM) stock

1. Download the daily IBM stocks price from 2003 to 2010 (inclusively). Fit a GARCH(1,1) model with normal errors. Is the model satisfactory? Make sure to check that the GARCH process is not integrated. Does the process display excess kurtosis, relative to that of the normal distribution (for which $\kappa = 3$).
2. If the errors are not normally distributed, experiment with a heavy-tailed distribution. Assess the adequacy of the latter using a QQ-plot.
3. If your returns are asymmetric, try a GARCH-type model that includes a leverage term. Is the fit better?

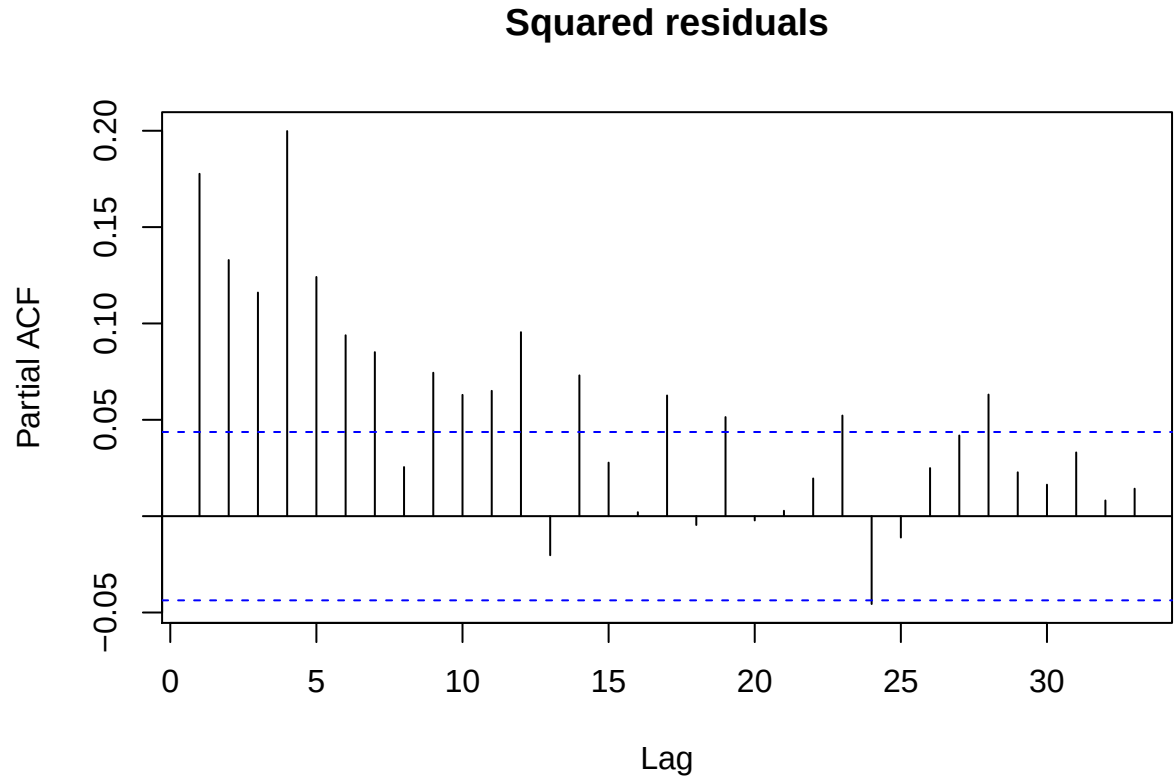
```
library(rugarch)
library(tseries)
ibm = get.hist.quote(instrument = "IBM", quote = "Adj", provider = c("yahoo"),
    method = NULL, start = "2003-01-01", end = "2010-12-31", compression = "d",
    retclass = c("zoo"), quiet = FALSE, drop = FALSE)
```

```
time series starts 2003-01-02
time series ends   2010-12-30
```

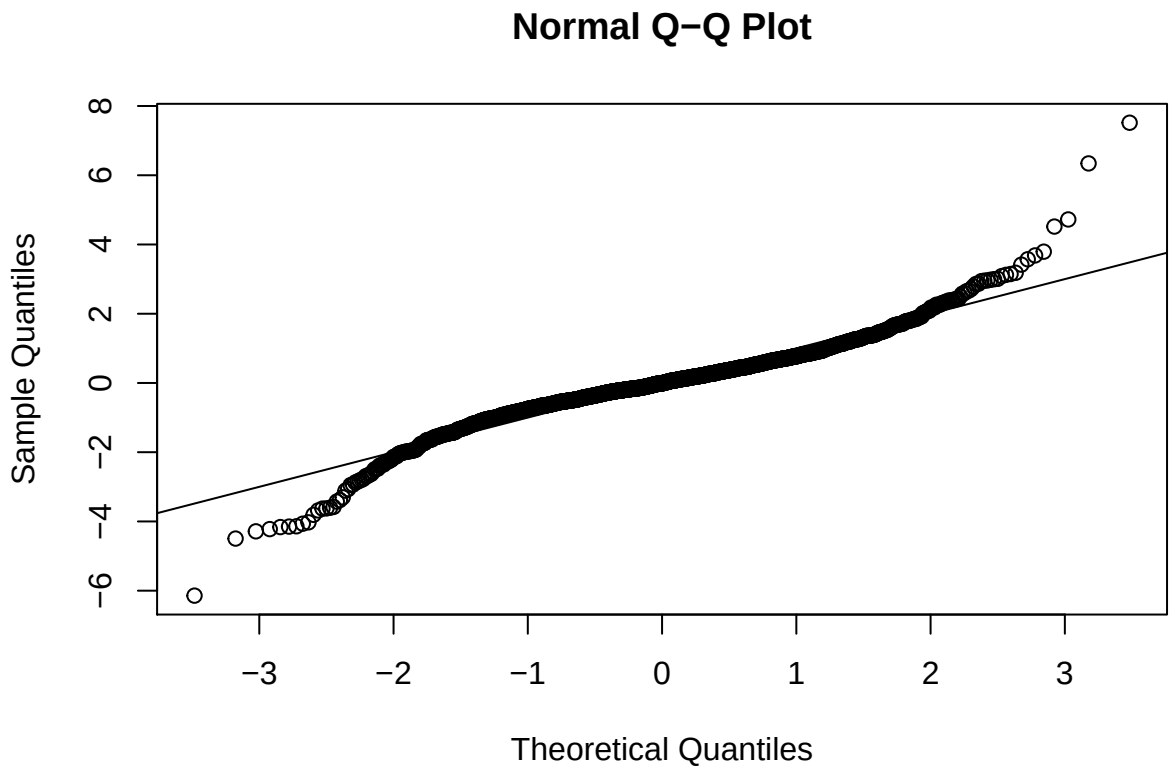
```
ibm_lret <- 100 * (log(as.vector(ibm[-1, ])) - log(as.vector(ibm[-length(ibm)])))

TSA::acf(I(residuals(lm(ibm_lret ~ 1))^2), main = "Squared residuals")
```



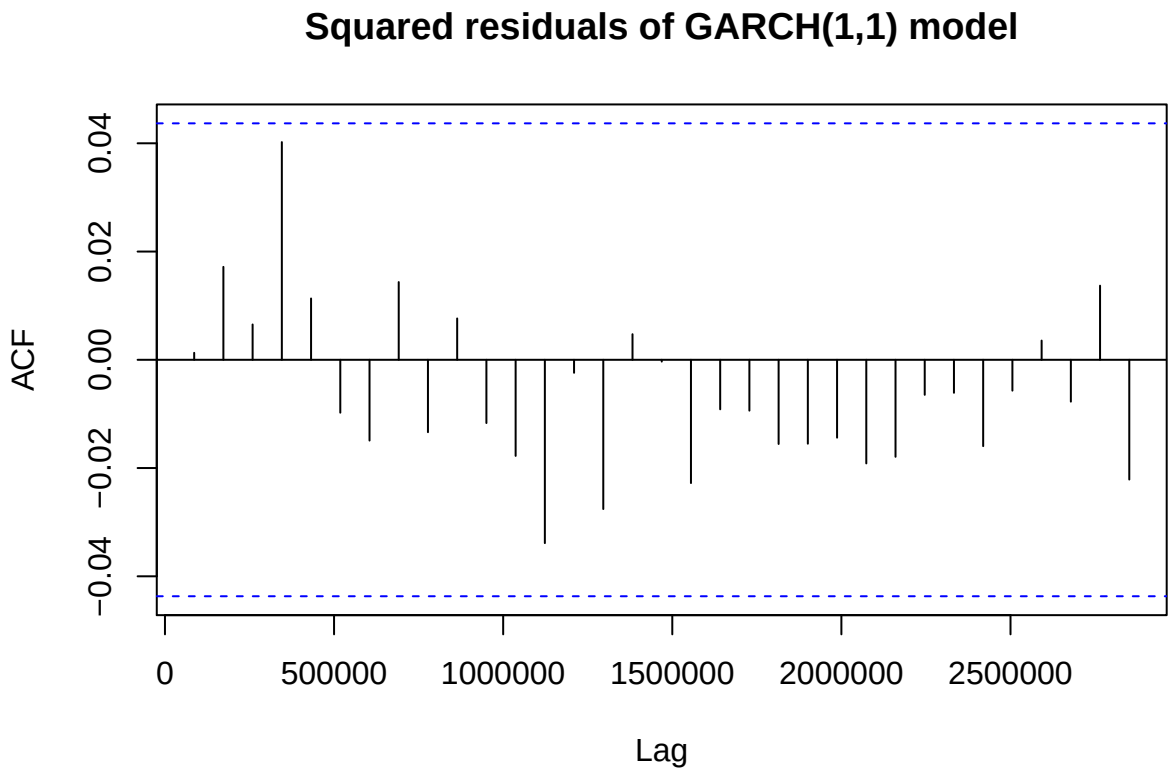

```
pacf(I(residuals(lm(ibm_lret ~ 1))^2), main = "Squared residuals")
```



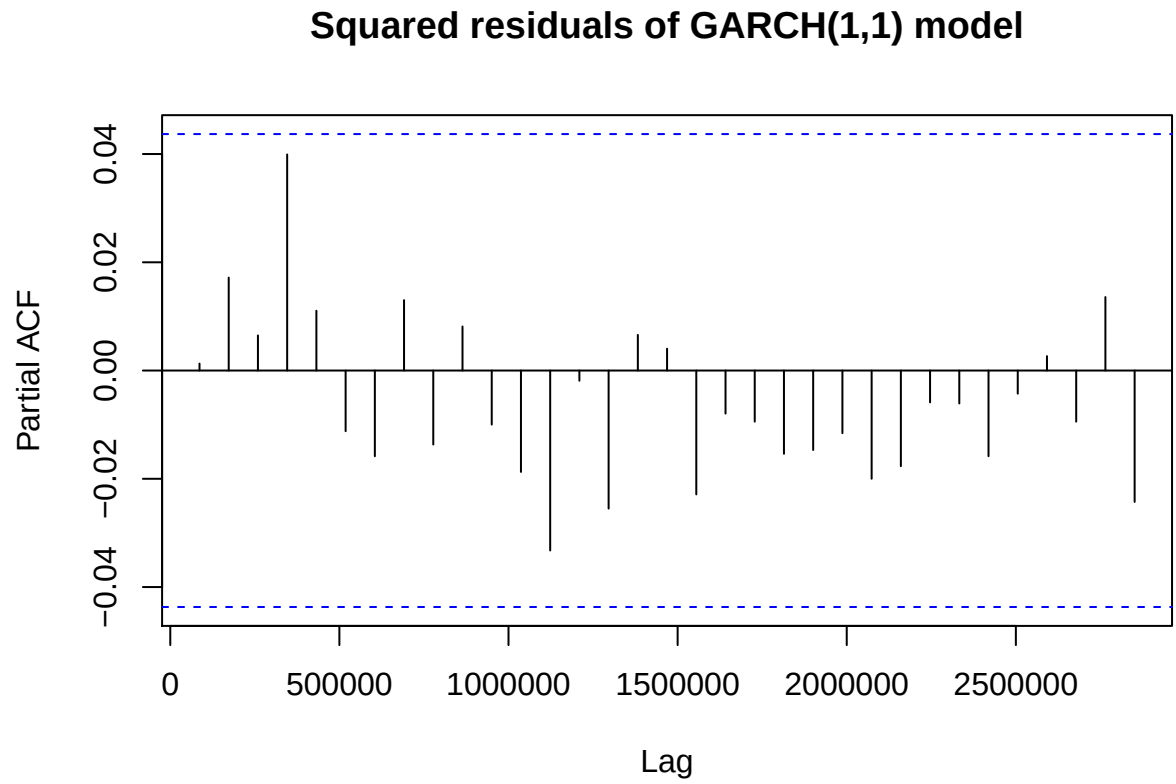
```
garch11_model <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1,
    1)), mean.model = list(include.mean = TRUE), distribution.model = "norm")
# Model fitting
garch11_fit <- ugarchfit(spec = garch11_model, data = ibm_lret, solver = "nloptr",
    solver.control = list(solver = 9))
qqnorm(scale(residuals(garch11_fit)), pty = "s")
abline(a = 0, b = 1)
```




```
sres <- residuals(garch11_fit)/sigma(garch11_fit)
# residuals(garch11_fit, standardize = TRUE) sres^2 is equivalent to
# residuals(garch11_fit)^2/garch11_fit@fit$var
TSA::acf(sres^2, main = "Squared residuals of GARCH(1,1) model")
```




```
pacf(sres^2, main = "Squared residuals of GARCH(1,1) model")
```



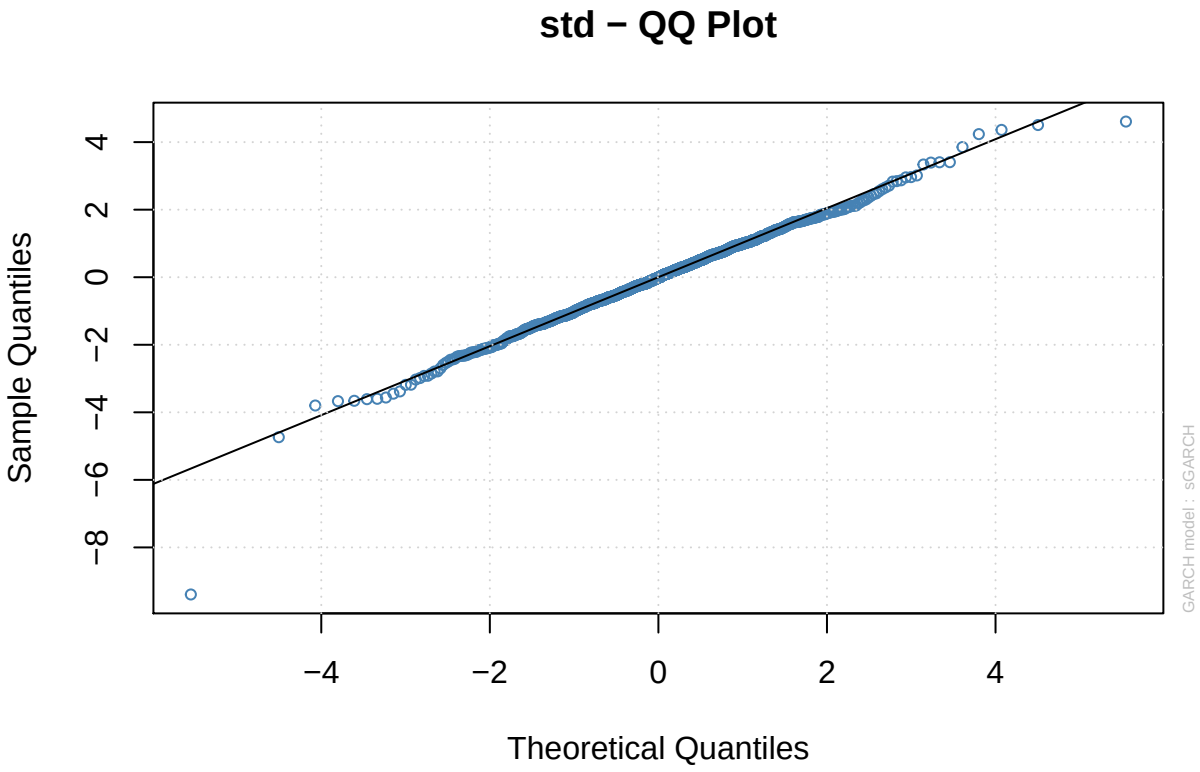
```
# Parameters of the GARCH(1,1)
alpha1 <- garch11_fit@model$pars["alpha1", 1]
beta1 <- garch11_fit@model$pars["beta1", 1]
```

If one calls `print(garch11_fit)`, we immediatly see that we have an IGARCH as $\alpha_1 + \beta_1 = 1$. The fact that β_1 is close to one is typical of heavy-tailed financial returns. The kurtosis is $\kappa = 3(1+\alpha_1+\beta_1)(1-\alpha_1-\beta_1)/(1-\beta_1^2-2\alpha_1\beta_1-3\alpha_1^2)$ if $3\alpha_1^2+2\alpha_1\beta_1+\beta_1^2<1$, provided the fourth moment exist (when $0 \leq \alpha_1+\beta_1<1$). A shock at one time becomes permanent, the conditional variance is not stationary. We attempt to fit a different model, a GARCH(a,b). We need $\alpha_1+\cdots+\alpha_a+\beta_1+\cdots+\beta_b<1$ for stationarity.

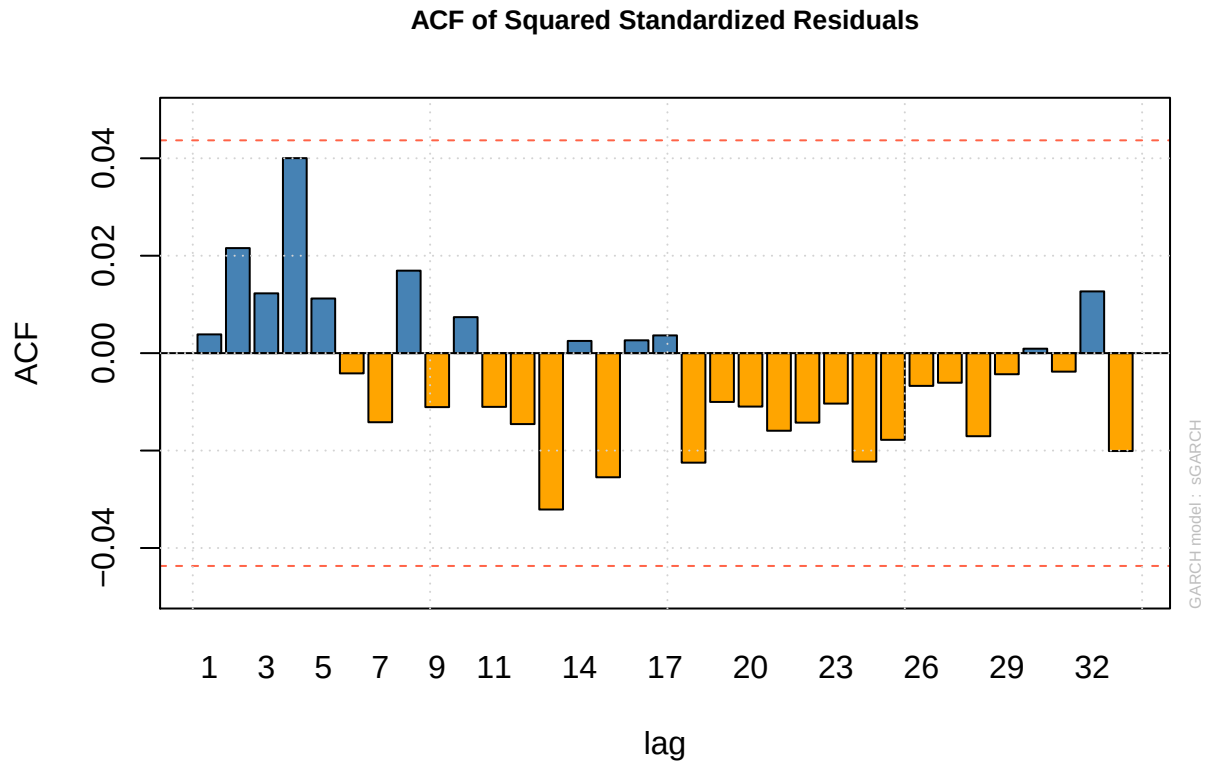
```
library(rugarch)
# Higher order needed (?), also more heavy-tailed innovations
garch21_model <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(2,
    1)), mean.model = list(include.mean = TRUE), distribution.model = "std")
garch21_fit <- ugarchfit(spec = garch21_model, data = ibm_lret, solver = "nloptr",
    solver.control = list(solver = 9))
# Is the conditional variance model stationary?
garch21_fit@model$pars["alpha1", 1] + garch21_fit@model$pars["alpha2", 1] +
    garch21_fit@model$pars["beta1", 1]
```

```
[1] 0.9932767
```

```
# Yes
plot(garch21_fit, which = 9)
```




```
plot(garch21_fit, which = 11)
```




```
# TSA::acf(c((residuals(garch21_fit)/sigma(garch21_fit))^2), main = 'Scaled
# residuals') pacf(c((residuals(garch21_fit)/sigma(garch21_fit))^2), main =
# 'Scaled residuals')

gjrgarch_model <- ugarchspec(variance.model = list(model = "gjrGARCH", garchOrder = c(1,
    1)), mean.model = list(include.mean = TRUE), distribution.model = "std")
gjrgarch_fit <- ugarchfit(spec = gjrgarch_model, data = ibm_lret, solver = "nloptr",
    solver.control = list(solver = 9))

# Not much improvement (if any) - may be visible however in the VaR
# forecasts left as an exercice!
```

Chapter 4

Spectral analysis and filtering

The following section is adapted from Percival and Walden *Spectral Analysis for physical applications*, CUP (1993). I will try to extract the most relevant notions so you can understand in greater details nonparametric spectral estimation, the periodogram, tapering and smoothing, and testing procedures.

As Percival and Walden say: "Spectral analysis is an analysis of variance".

Roughly speaking, the aim of spectral analysis is to break the variability of the series into different contributions corresponding to a frequency.

We shall analyze stationary sequences $\boldsymbol{X} \in \mathbb{R}^n$ thought to originate from a stationary process $\{\boldsymbol{X}_t\}$, $t \in \mathbb{Z}$. All the information is contained in the spectral density function assuming the latter exists.

4.1 Nonparametric spectral estimation

In the course, the sampling interval Δ_t is one, so that the Nyquist frequency $\omega_{\mathsf{N}} \equiv 1/(2\Delta_t)$ is $1/2$.

The spectrum is the Fourier transform of the autocovariance, viz.

$$f(\omega) = \sum_{h=-\infty}^{\infty} \gamma_h \exp(-2\pi i h\omega).$$

We thus first must estimate $\gamma_h = \mathrm{E}\{(X_{t+h} - \mu)(X_t - \mu)\}$. We typically use the sample mean $\overline{X}$. The sample autocovariance is

$$\widehat{\gamma}_h = \frac{1}{n} \sum_{t=1}^{n-|h|} (X_{t+|h|} - \overline{X})(X_t - \overline{X}) = \left(1 - \frac{|h|}{n}\right) \widehat{\gamma}_h^{(u)}$$

where

$$\widehat{\gamma}_h^{(u)} = \frac{1}{n-|h|} \sum_{t=1}^{n-|h|} (X_{t+|h|} - \overline{X})(X_t - \overline{X})$$

is the usual unbiased estimator of γ_h.

If we replaced the sample mean estimate $\overline{X}$ by μ, then

$$\mathrm{E}(\widehat{\gamma}_h) = \left(1 - \frac{|h|}{n}\right) \gamma_h.$$

The estimator $\widehat{\gamma}_h$ is thus biased, but

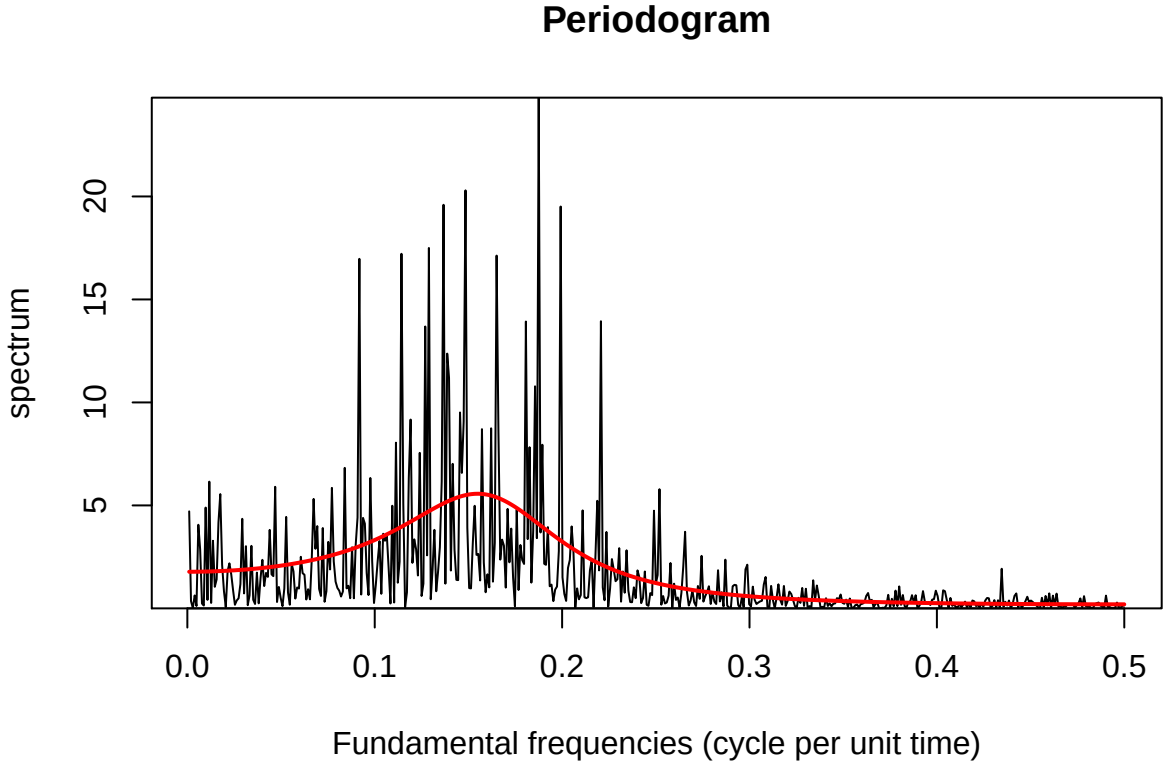
1. it has typically lower mean squared error than $\widehat{\gamma}_h^{(u)}$,
2. it gives a positive definite sequence.
3. if $\gamma_h \to 0$ as $h \to \infty$, then $\widehat{\gamma}_h$ is guaranteed to go to zero as $h \to n$.

The discrete time autocovariance are related to the estimates of the spectrum at continuous frequency,

$$\widehat{\gamma}_h = \int_{-1/2}^{1/2} \widehat{f}(\omega) \exp(2\pi i h\omega) \mathrm{d}\omega.$$

The estimates $\{\widehat{\gamma}_k : k = -n+1, \ldots, n-1\}$ and $\{\widehat{f}(\omega_k) : \omega_k = k/(2n-1), k = -n+1, \ldots, n-1\}$ thus form a Fourier transform pair.

Let us look at the periodogram estimate $I(\omega)$ of a simulated stationary autoregressive process with $n = 1000$. We can compare it to the true spectrum of the AR(2) process with $\phi_1 = 0.75$, $\phi_2 = -0.5$ and $\sigma^2 = 1$.



What we notice right away is how noisy the periodogram estimate is. The truth is that the periodogram is a biased and inconsistent estimator: we estimate a frequency based on 2 observations, so even as $n \to \infty$, the estimates remain extremely variable. The estimates for distinct frequencies are however almost uncorrelated provided the Fourier frequencies are far and that the bias is not too large.

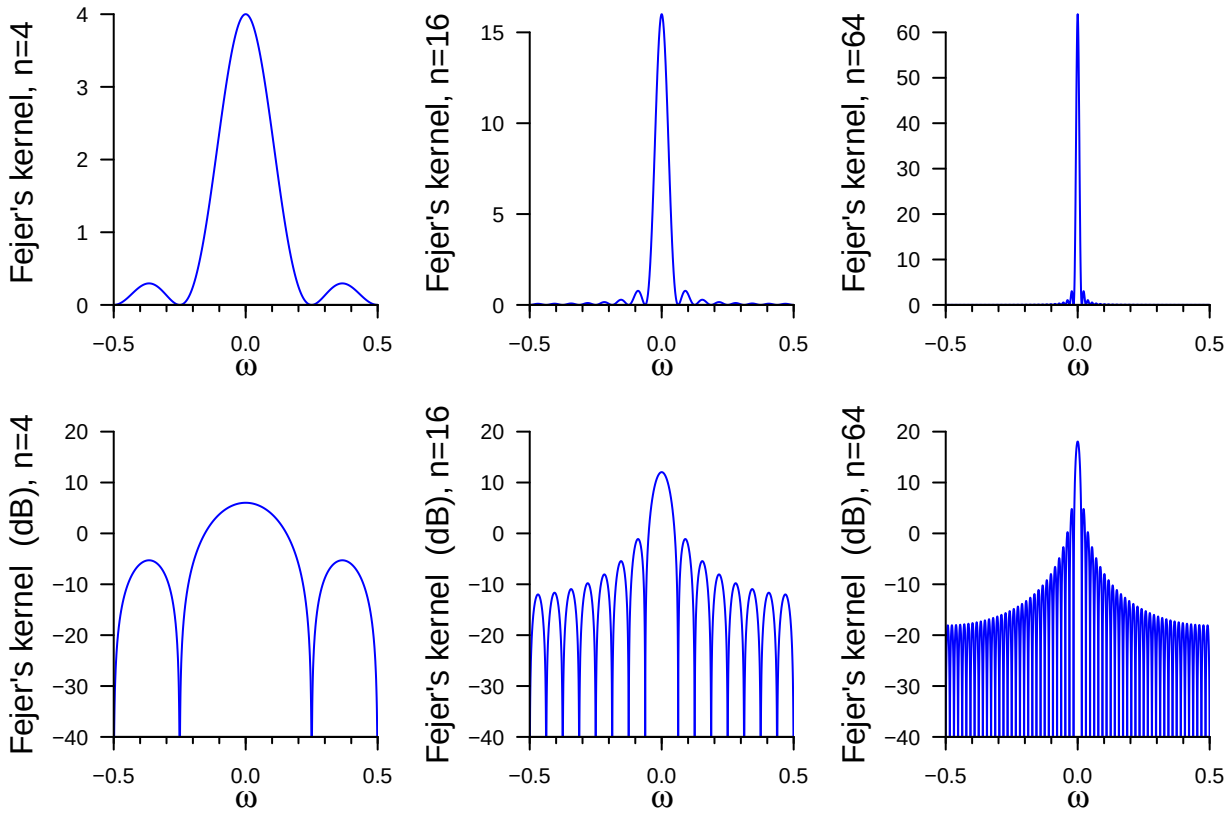
The expectation of the periodogram is

$$\begin{aligned} \mathrm{E}\big(I(\omega_j)\big) &= \sum_{h=-n+1}^{n-1} \left(1 - \frac{|h|}{n}\right) \gamma_h \exp(-2\pi i \omega h) \\ &= \int_{-1/2}^{1/2} n\mathcal{D}_n^2(\omega_j - \omega) f(\omega) \mathrm{d}\omega \end{aligned}$$

and $n\mathcal{D}^2(\omega) \equiv W_n(\omega)$ (in the course notation) is the rescaled squared Dirichlet kernel, also known as Fejér's kernel,

$$\mathcal{F}(\omega) = n\mathcal{D}^2(\omega) = \frac{\sin^2(n\pi\omega)}{n\sin^2(\pi\omega)}.$$

The Fejér kernel concentrates as $n \to \infty$ and behaves asymptotically like a δ function. However, for finite n, it has marked side lobes — these are more visible on the log scale or the decibel scale ($10\log_{10}(\cdot)$). One can view the effect of increased sample size by plotting the Fejér kernel for $n = 4, 16, 64$.

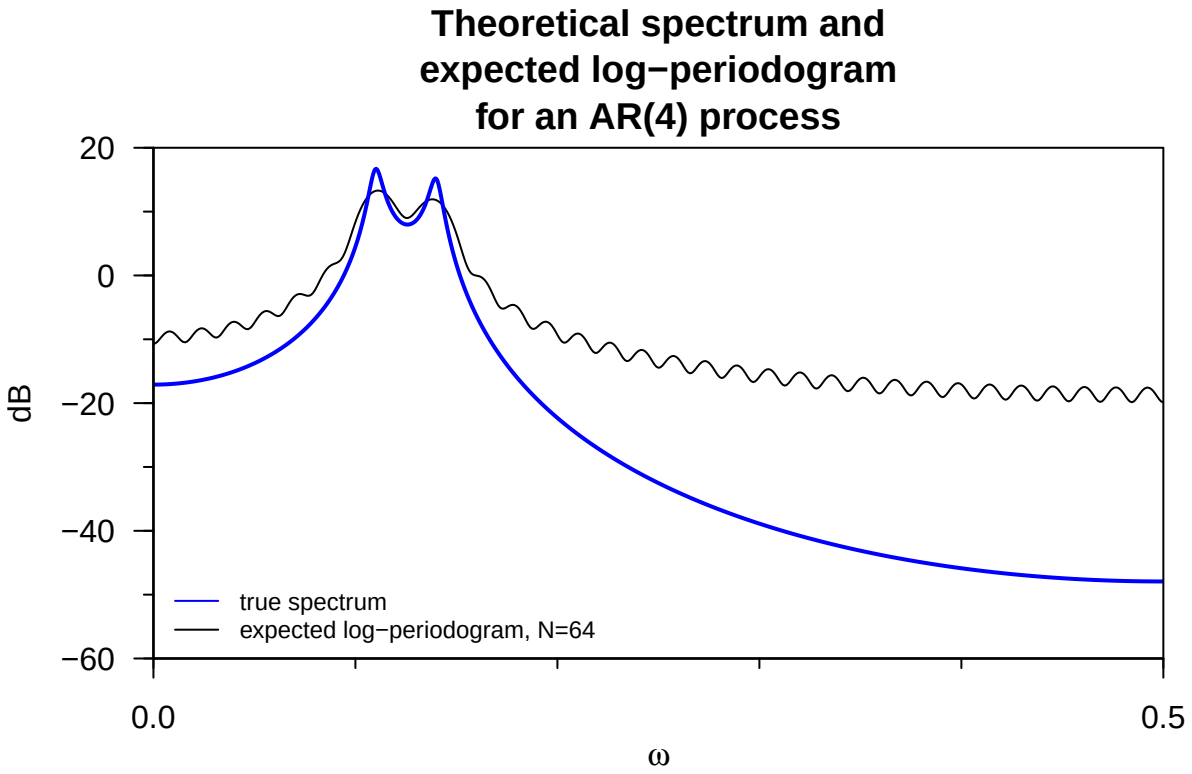


The periodogram is asymptotically unbiased, but the bias largely depends on the dynamic range. The latter is defined as

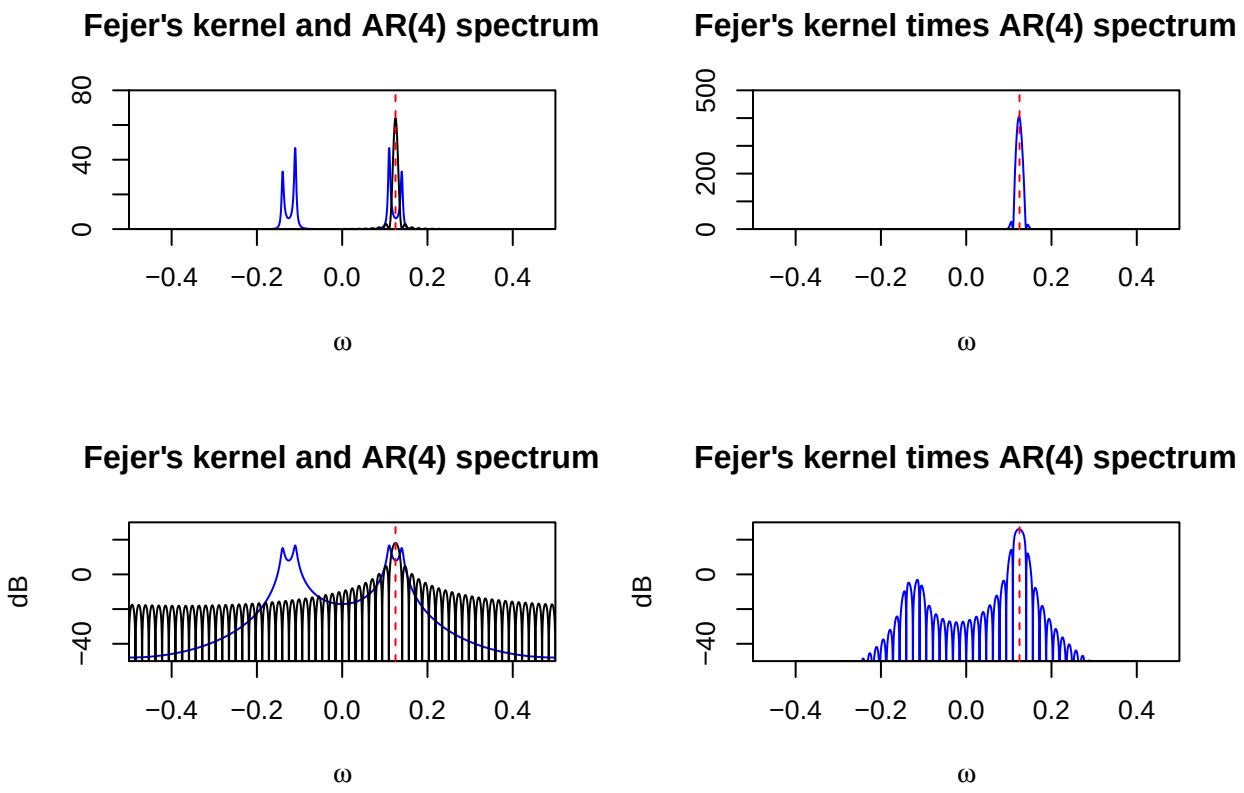
$$d_r = 10\log_{10}\left(\frac{\max_\omega f(\omega)}{\min_\omega f(\omega)}\right).$$

For white noise, this is zero, so the bias is barely visible even in small samples. However, for processes with high dynamic range, it can be significant even (with n as large as a million!)

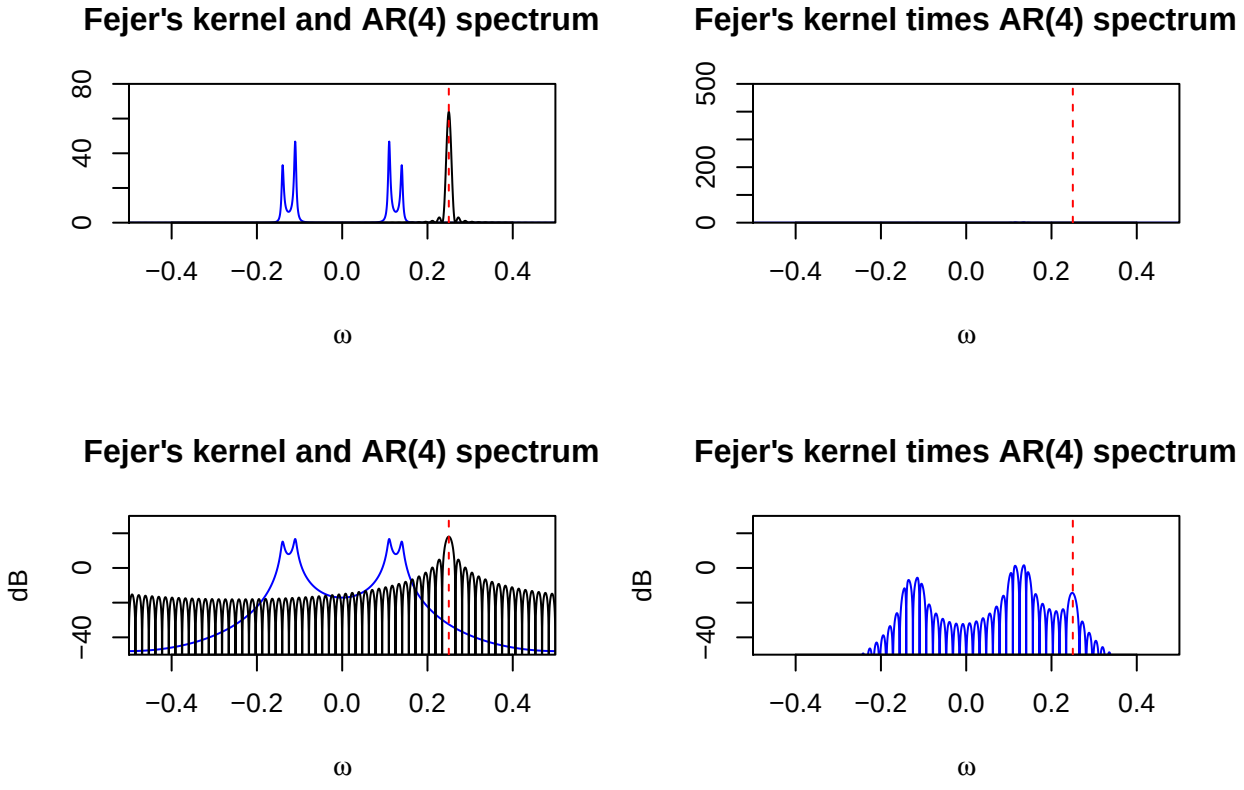
The following example from Percival and Walden illustrates both the bias and the leakage, plus the effect of the sidelobes. The plot shows the true spectrum and the expected values for the periodogram ordinates.



How did these lobes appear? Let's look at what the Fejér kernel does when convolved with the spectrum. Again, the effect is more apparent on the decibel scale. We focus on the frequency $\omega = 1/8$, which corresponds to the center between the two peaks. The left panels show the Fejér kernel centered at $\omega = 1/8$, while the right one shows the product of the kernel with the spectrum. While we typically only plot $f(\omega)$ on $(0, 1/2)$ because of the symmetry, the $f(\omega)$ contributes for $(-1/2, 0)$ as well so it is displayed. The contribution is the sum of the product of the Fejér kernel at $\omega = 1/8$ with the true spectrum.



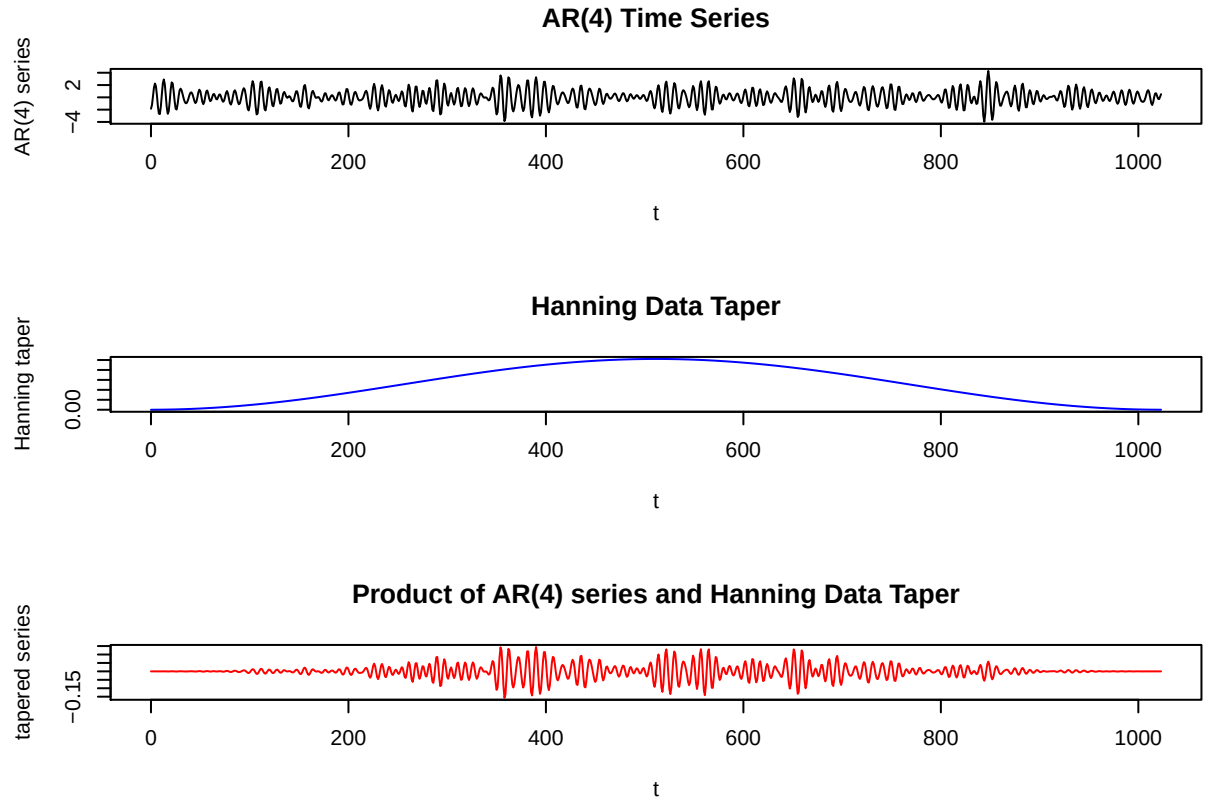
Same thing, but this time for the frequency $\omega = 1/4$.



For our AR(4) process with $n = 64$ observations, leakage will happen mostly at high frequencies, $\omega > 0.3$, where the true spectrum is nearly flat.

4.1.1 Tapering

The goal of tapering is to reduce the bias by reducing the magnitude of sidelobs in the Fejér kernel. In order to do so, we multiply the data by a taper h_t, to form a new series $X_t h_t$ on which we compute the usual periodogram. The resulting estimator is a direct estimate, denoted $I^d(\omega)$.



One can show that

$$\mathrm{E}\left(I^d(\omega_j)\right) = \int_{-1/2}^{1/2} W_n(\omega_j - \omega) f(\omega) \mathrm{d}\omega$$

where the spectral window $W_n(\omega)$, in the course notation, is

$$W_n(\omega) = \frac{1}{n}\left|\sum_{t=0}^{n-1} h_t \exp(-2\pi i \omega t)\right|^2$$

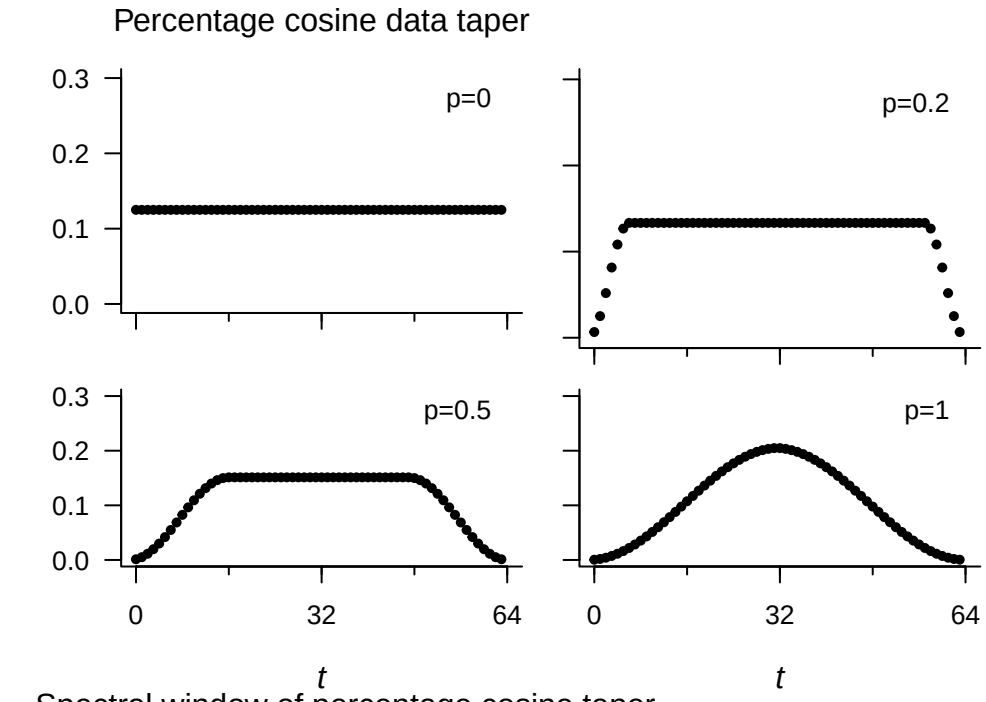
A rectangular data taper in which $h_t = 1$ gives back Fejer's kernel $\mathcal{F}$ and the periodogram. We will thus consider other tapers, $\mathcal{H}$ say, which are such that h_t progressively goes to zero.

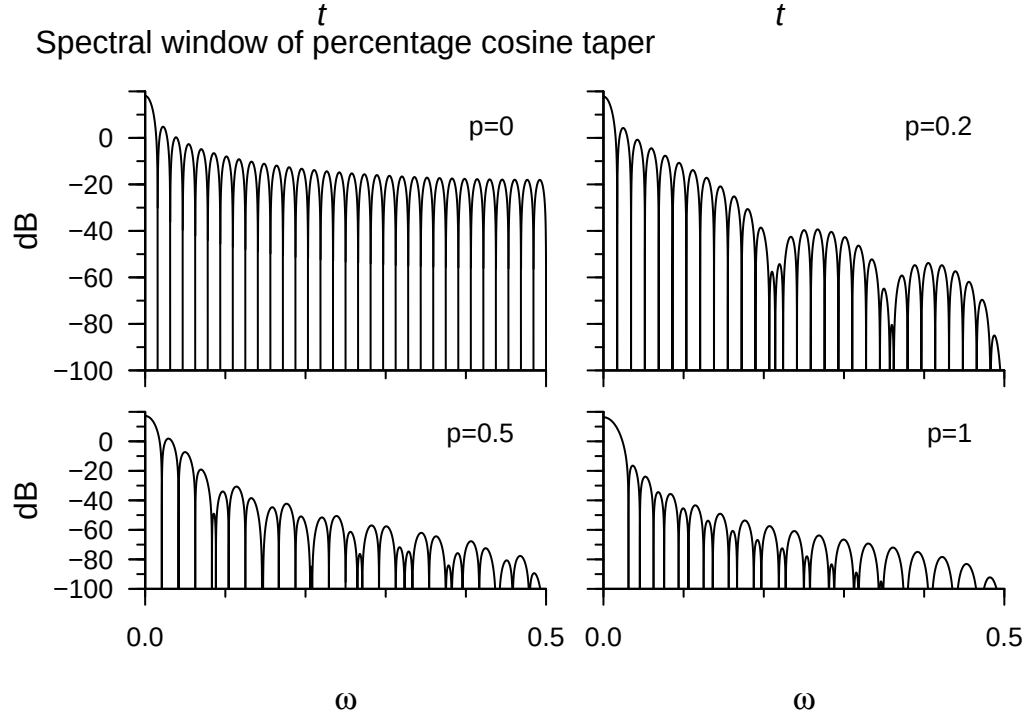
To reduce bias due to leakage, we want $\mathcal{H}$ to have smaller sidelobes than $\mathcal{F}$. There is no free lunch: such kernels $\mathcal{H}$ typically have bigger main lobes and thus smears out the peaks, introducing a new source of bias. As such, tapering is most useful for processes with large dynamic ranges. Because tapering discards information, the variance of the estimates also increases.

The next plots show the data taper sequences h_t for $t = 1, \ldots, n$ and the corresponding spectral window $\mathcal{H}$ function on the decibel scale. It corresponds to an mix between the Hanning taper, which ressembles a bell curve, and the rectangular weights. The default taper in R smooths the first and last p percentage of the time points.

$$h_t = \begin{cases} \frac{1}{2}\left(1-\cos\left(\frac{\pi t}{np}\right)\right), & \text{if } 1 \le t < np \\ 1, & \text{if } np \le t \le n(1-p) \\ \frac{1}{2}\left(1-\cos\left(\frac{\pi(n+1-t)}{np}\right)\right), & \text{if } n(1-p) < t \le n. \end{cases}$$

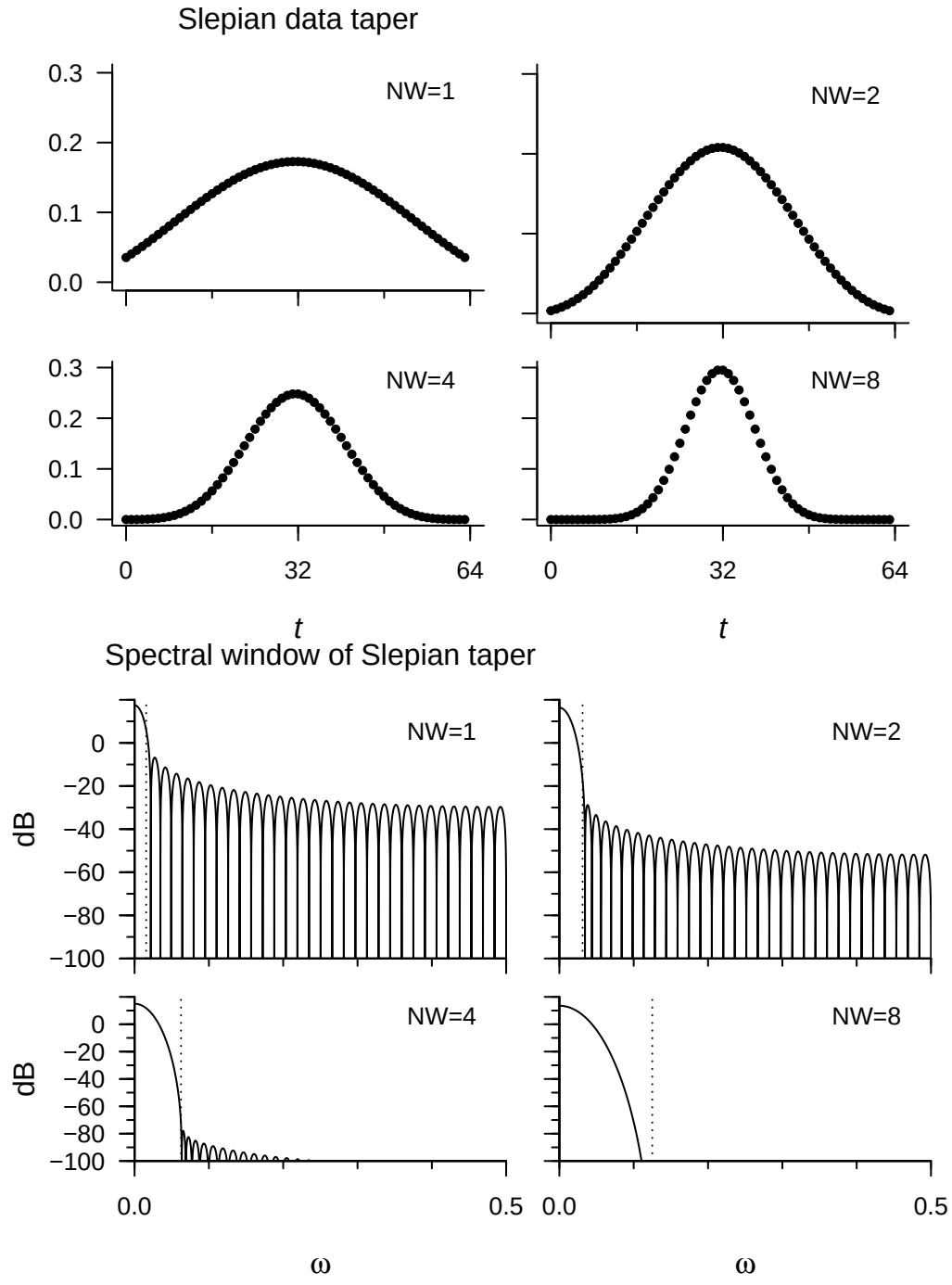
The proportion $p = 0$ gives back the raw periodogram, while $p = 1$ gives the Hanning taper. The tapers and spectral windows (for the portion $(0, 1/2)$ are shown here for $p = 0, 0.2, 0.5, 1$.



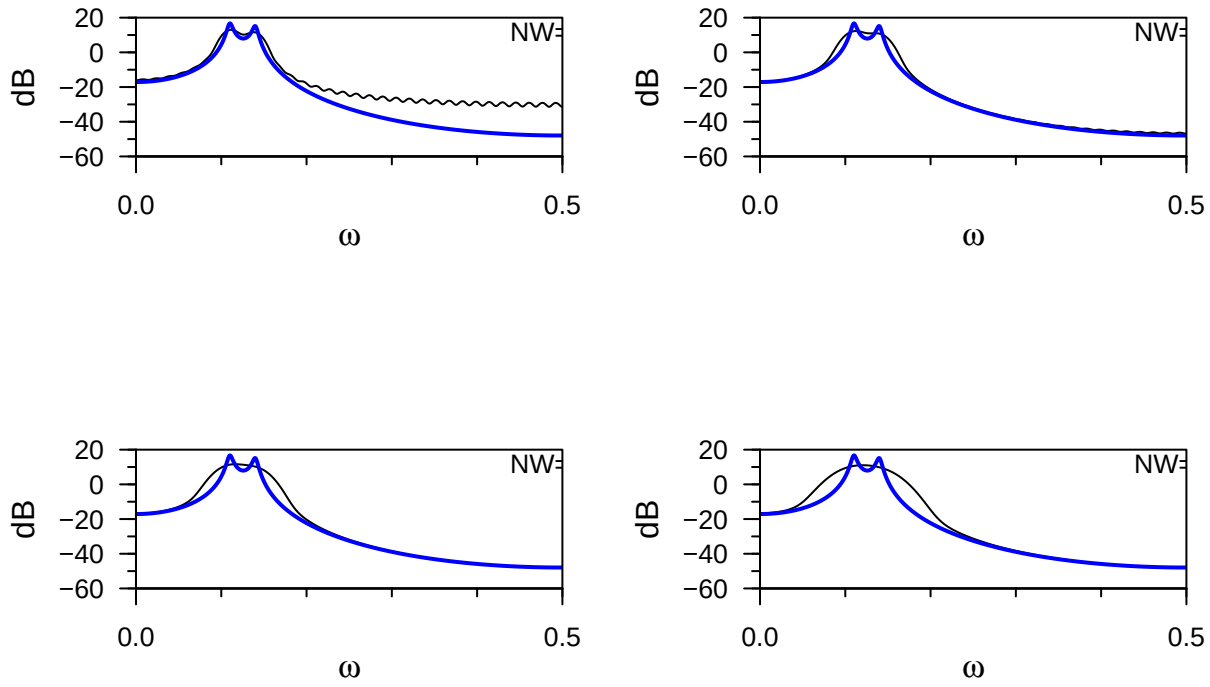


As we increase the tapering, the sidelobes of become progressively smaller, but the width of the main lobe increases. This we will illustrate with a special taper, for which the phenomenon is most visible.

The Slepian tapers are special tapers that minimize the energy in the side lob. They are the filters whose transfer function is maximized in a specified range $|\omega| < W$. They are oftentime parametrized in terms of nW, and $nW = 1$ gives back (roughly speaking) the Fejér kernel, while $nW = 2$ is closer to the Hanning taper.



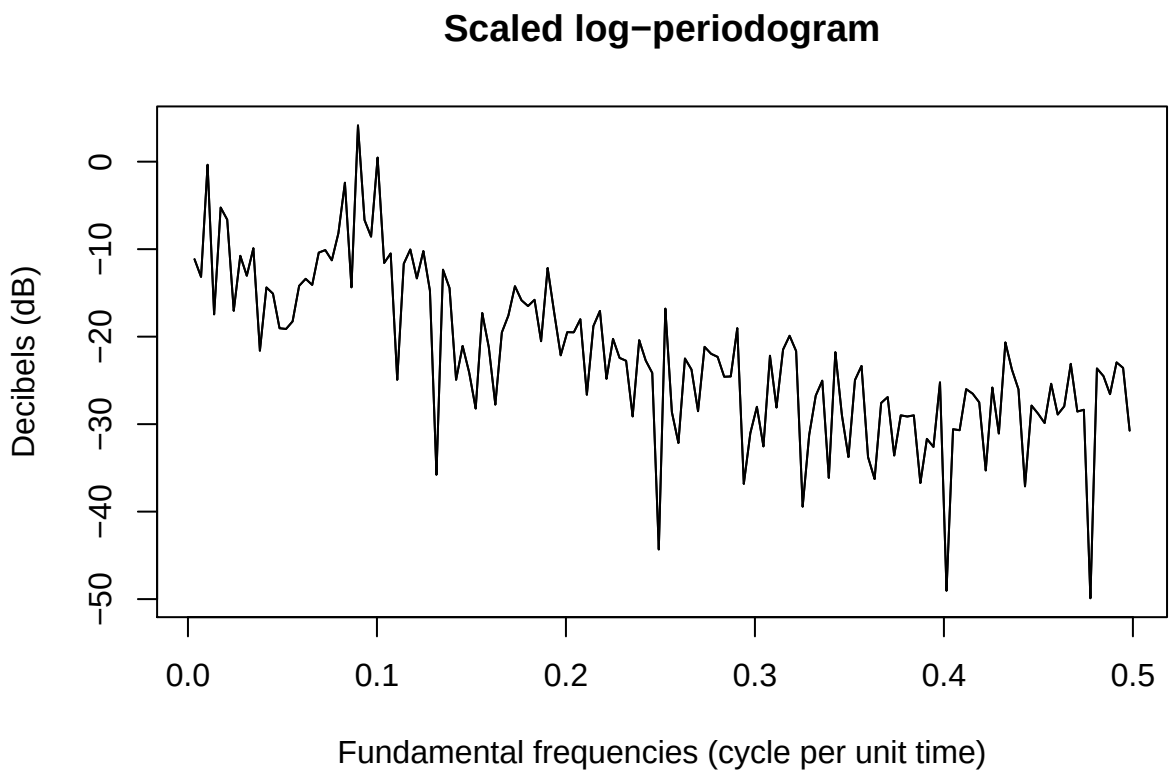
What happens when we overtaper a series? The following plot shows the expected value of the direct estimate of the spectrum based on 64 observations (!) compared to the true estimate. The underlying spectrum is that of an AR(4) series.



4.1.2 A data example

We now illustrate the calculation of the raw periodogram on the sunspot dataset.

```
source("http://faculty.washington.edu/dbp/s520/R-code/tapers.R")
sunspot = c(scale(sqrt(sunspot.year), scale = FALSE))  #Square root transformed sunspot, centered
n <- length(sunspot)
nj <- floor(n/2)
om <- (1:nj)/n
# Calculate manually the periodogram using the formulas for the regression
# coefficients
a <- b <- vector(mode = "numeric", length = nj)
for (j in 1:nj) {
    a[j] <- sum(sapply(1:n, function(t) {
        sunspot[t] * cos(2 * pi * t * j/n)
    }))
    b[j] <- sum(sapply(1:n, function(t) {
        sunspot[t] * sin(2 * pi * t * j/n)
    }))
}
# Plot the log-periodogram
plot(om, 10 * log10(2/n^2 * (a^2 + b^2)), main = "Scaled log-periodogram", xlab = "Fundamental frequenc
    ylab = "Decibels (dB)", type = "l")
# Repeat the calculations, this time using the FFT
lines(om, 10 * log10(Per <- 2 * (Mod(fft(sunspot))[2:(nj + 1)]/n)^2))  #lines overlayed
```

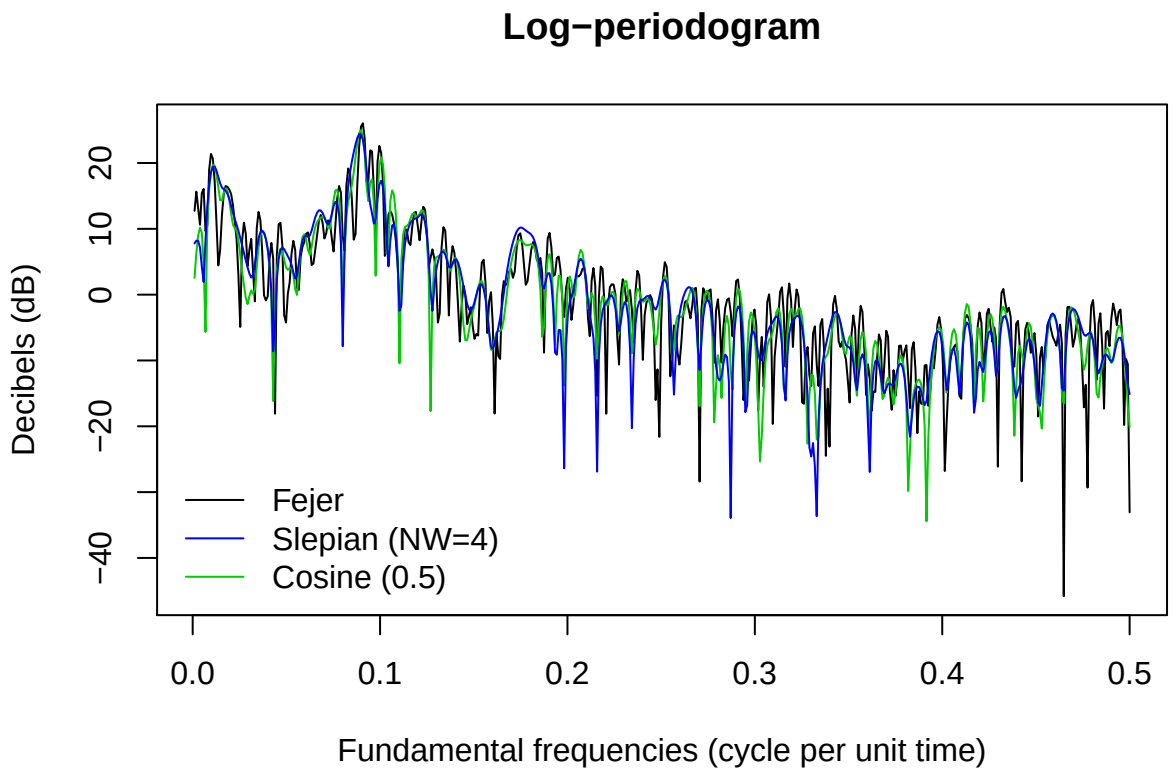


The two definitions are thus identical, but one must be careful with the different scalings employed in different references.

It is customary to include trailing zeros in the series to make the latter divisible by a low prime, potentially doubling its length. Padding with zero does not affect the properties of the estimator, but rather the frequency at which the estimates are computed. Padding is used to make computations faster.

We illustrate zero-padding and tapering in the next plots. We abstain from now from smoothing.

```
# plot((1:nj)/n, Per, type='l', xlab = 'Frequency', ylab = 'Spectral
# density', main = 'Periodogram') Zero padding
M <- 2 * nextn(n, factors = 2)  #1024, next power of 2 greater than sample size
# calculate the periodogram with the FFT with zero padding
ffreq <- seq(1/M, 0.5, by = 1/M)
spec_pgram <- abs(fft(c(sunspot/sqrt(n), rep(0, M - n)))[2:(M/2 + 1)])^2  #square modulus of FFT data,
plot(ffreq, 10 * log10(spec_pgram), xlab = "Fundamental frequencies (cycle per unit time)",
    ylab = "Decibels (dB)", type = "l", main = "Log-periodogram")
# Tapering
tapc_sunspot <- cosine.taper(N = n, p = 0.5) * sunspot
taps_sunspot <- slepian.taper(N = n, NW = 2) * sunspot
spec_tap_cos <- abs(fft(c(tapc_sunspot, rep(0, M - n)))[2:(M/2 + 1)])^2
spec_tap_sle <- abs(fft(c(taps_sunspot, rep(0, M - n)))[2:(M/2 + 1)])^2
lines(ffreq, 10 * log10(spec_tap_cos), col = 3)
lines(ffreq, 10 * log10(spec_tap_sle), col = 4)
legend(x = "bottomleft", col = c(1, 4, 3), lty = c(1, 1, 1), legend = c("Fejer",
    "Slepian (NW=4)", "Cosine (0.5)"), bty = "n")
```



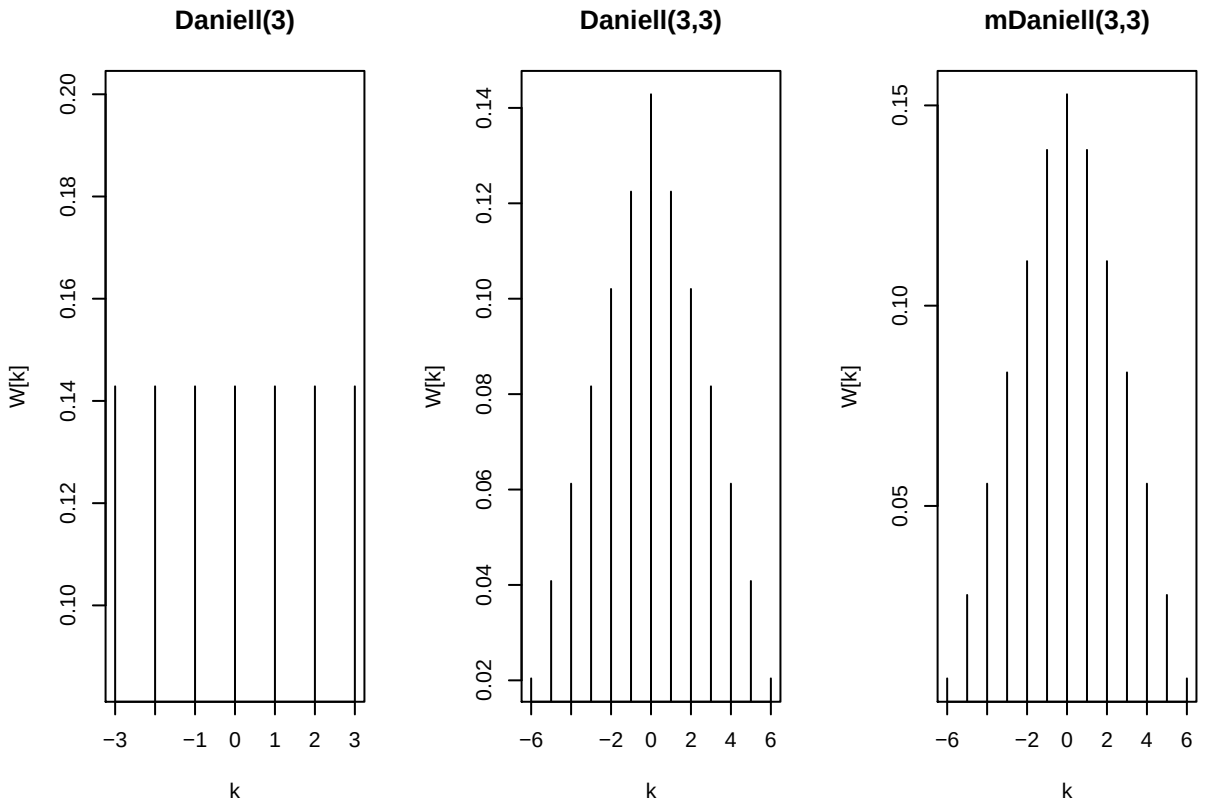
Since the dynamic range is low, tapering has little effect here.

4.1.3 Smoothing

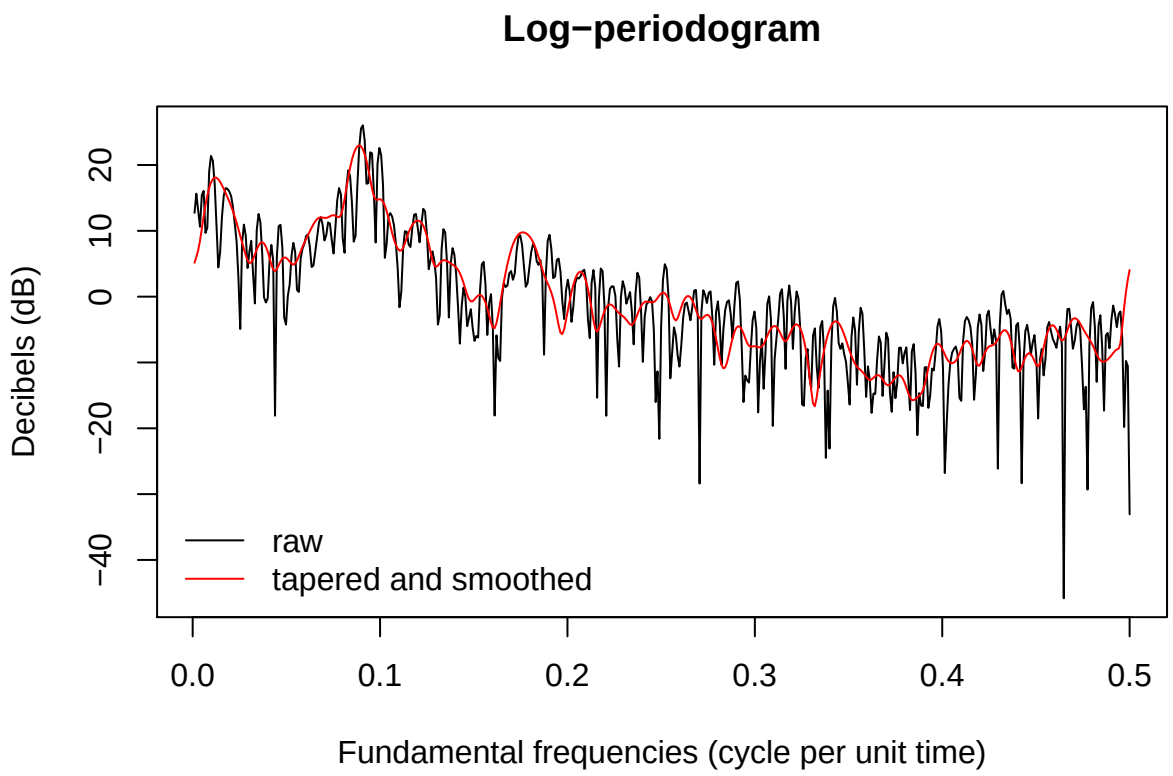
Smoothing is the last modification we shall cover. The goal here is to convolve the periodogram estimate for the tapered data with a kernel. We typically consider linear filters that reproduce simple moving average. The effective range of the smoother depends on a bandwidth parameter, but the latter is model dependent.

The smoothing illustrated below is performed using modified Daniell smoothers, which are simply moving averages that give half weight to the end values of the span. The simple Daniell filter gives equal weight, but it can be applied recursively. Increasing `spans` smooths the raw periodogram plotted in the first call to `spectrum`. The spans should be odd integers, and smoother results are obtained if they are different and at least two are used.

```
par(mfrow = c(1, 3))
plot(kernel("daniell", m = 3))
plot(kernel("daniell", m = c(3, 3)))
plot(kernel("modified.daniell", m = c(3, 3)))
```




```
ker <- kernel("modified.daniell", m = c(3, 3))
par(oldpar)
# Convolve the kernel with the periodogram
smooth_spec <- kernapply(spec_tap_sle, ker, circular = TRUE)
plot(ffreq, 10 * log10(spec_pgram), xlab = "Fundamental frequencies (cycle per unit time)",
    ylab = "Decibels (dB)", type = "l", main = "Log-periodogram")
lines(ffreq, 10 * log10(smooth_spec), col = 2)
legend(x = "bottomleft", col = c(1, 2), lty = c(1, 1), legend = c("raw", "tapered and smoothed"),
    bty = "n")
```



4.2 Summary of nonparametric spectral estimation

In summary, the different steps to undertake are, in order,

1. Differencing (or detrending) should be performed prior to any analysis. The series should be centered around zero by subtracting the mean from the data to remove the frequencies near $\omega = 0$, which would otherwise dominate.
2. For tapering, multiply centered observations by a data window. The cosine bell taper, also known as Hann taper, is the default choice in the R function `spectrum`.
3. Zero-pad to increase the resolution and so that the length of your time series is a highly composite number, typically $2^l > n$. If you want to convert your spectrum to an autocovariance, zero-padding helps avoid "circular autocorrelation". It also reduces the computation burden for the FFT calculation.
4. Smooth the periodogram. It amounts to convoluting the periodogram with a kernel. The amount of averaging should increase as $n \to \infty$. This reduces the variance, but introduces leakage. The amount of smoothing is controlled by the bandwidth parameter.

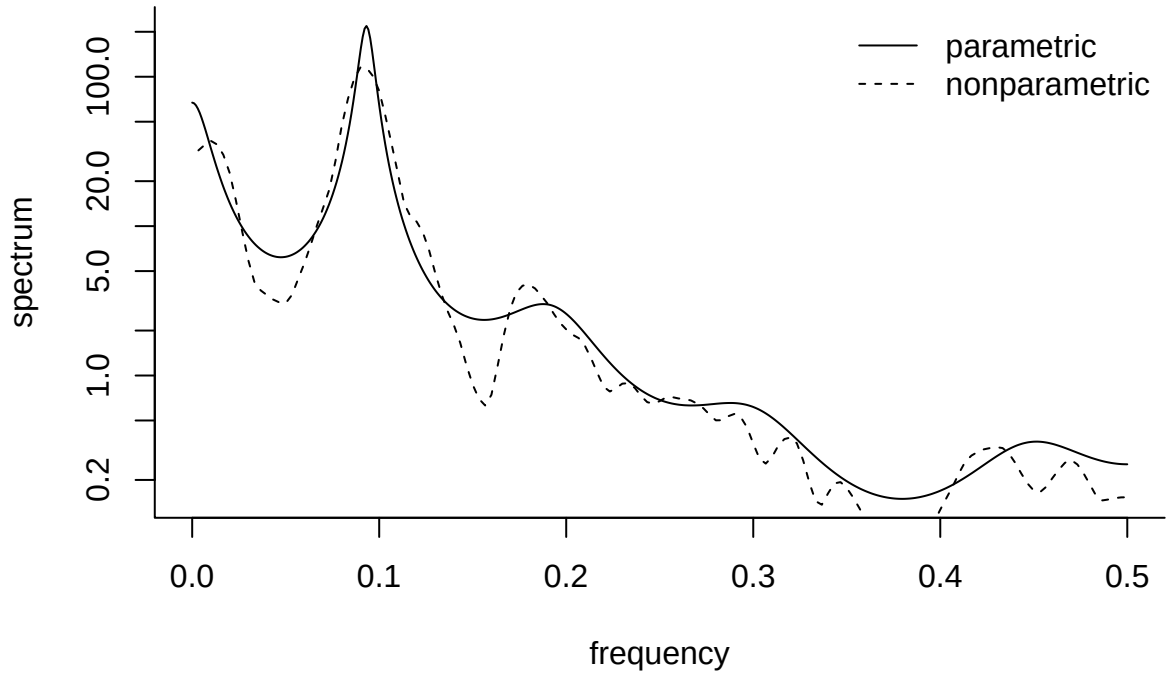
4.3 Spectral estimation in R

The workhorse for spectral estimation is the function `spectrum`, which calls `spec.pgram` in the background for nonparametric spectral estimation. It uses by default the modified Daniell's filters, whose argument are fixed via `spans`. The function uses the percentage cosine taper, with `taper=0.1` as default. The option `fast` is used for zero-padding. The function `TSA::spec` is equivalent to `spectrum`.

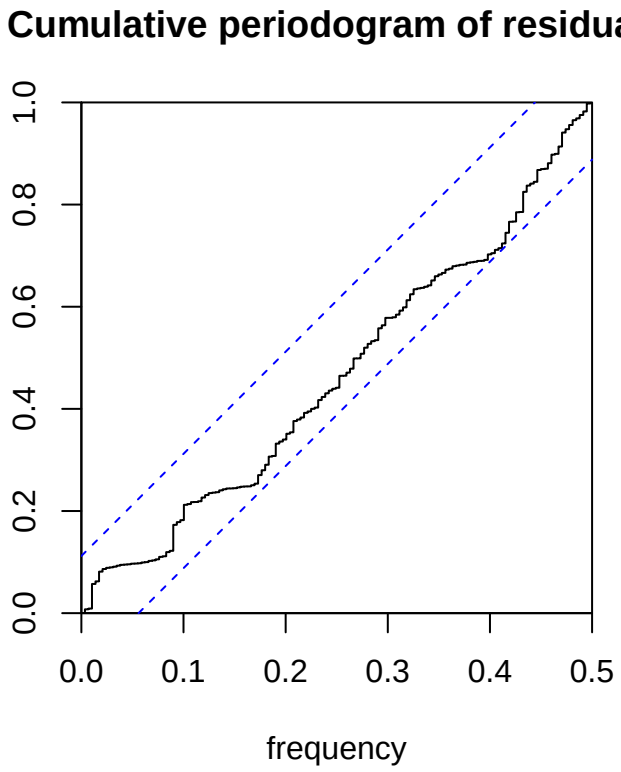
One can also resort to parametric estimation via the `spec.ar`. The package `astsa` has a function `astsa::arma.spec` to display the spectrum of a given ARMA model (see also `TSA::ARMAspec`).

Here is a simple example using the function `spectrum`. The last line performs the graphical test for white noise using the cumulative periodogram (with default tapering of 0.1). The confidence bands are derived using the asymptotic approximate χ^2 distribution. The height of the line to the right of the plot shows the width of a 95% confidence interval for the underlying function, and the width of its centre mark indicates the bandwidth parameter.

```
sun_ar = spec.ar(sunspot, plot = FALSE)  # parametric estimation based on AR model
# The latter's order is estimated using the Yule-Walker equations, with
# order selected by AIC
sun_np = spectrum(sunspot, spans = c(5, 5), plot = FALSE)  # nonparametric
plot(sun_ar$freq, sun_ar$spec, type = "l", log = "y", ylab = "spectrum", xlab = "frequency",
    bty = "l")
lines(sun_np$freq, sun_np$spec, lty = 2)
legend("topright", c("parametric", "nonparametric"), lty = 1:2, bty = "n")
```




```
cpgram(resid(arima(sunspot, c(2, 0, 0))), main = "Cumulative periodogram of residuals")
```



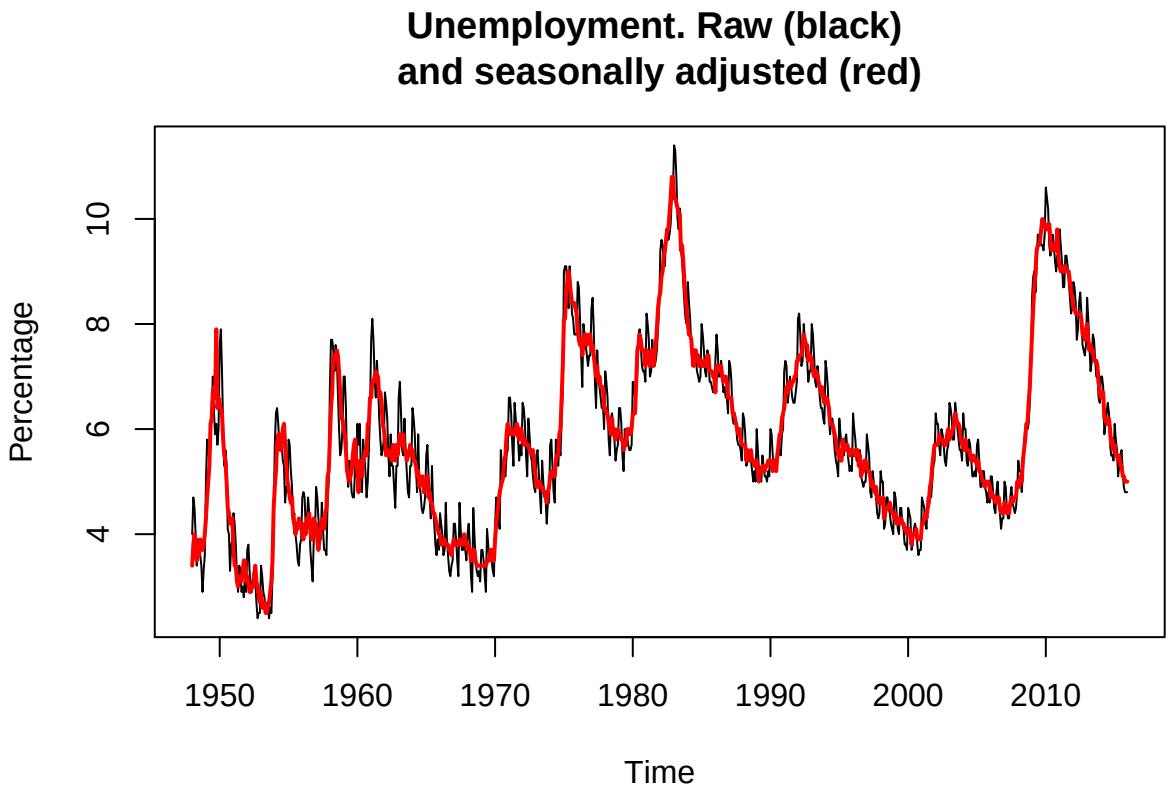
4.3.1 Smoothing and seasonally adjusted values

The following is taken from Edward Ionides course[1] and licensed under Creative Commons attribution-noncommercial license, http://creativecommons.org/licenses/by-nc/3.0/.

Suppose one is interested in long-term trend in unemployment in the US. The following series provide both raw and seasonally adjusted figures for monthly unemployment.

```
U1 <- read.csv(file = "https://raw.githubusercontent.com/ionides/531w16/master/notes08/unadjusted_unempl
    skip = 8, header = TRUE, sep = ",")

U2 <- read.csv(file = "https://raw.githubusercontent.com/ionides/531w16/master/notes08/adjusted_unemploy
    skip = 8, header = TRUE, sep = ",")

u1 <- ts(c(unlist(t(U1[, 2:13]))), start = c(1948, 1), frequency = 12)
plot(u1, ylab = "Percentage")
u2 <- ts(c(unlist(t(U2[, 2:13]))), start = c(1948, 1), frequency = 12)
lines(c(time(u2)), c(u2), col = 2, lwd = 2)
title("Unemployment. Raw (black) \nand seasonally adjusted (red)")
```

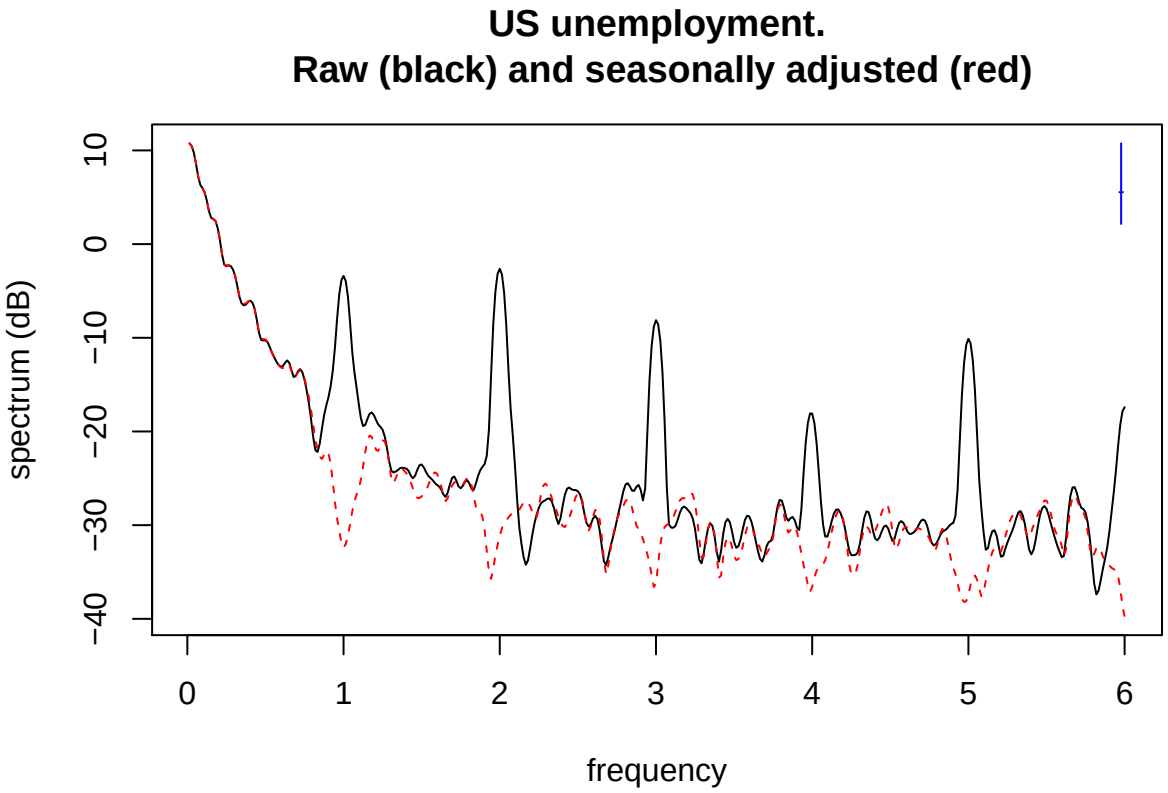
[1] http://ionides.github.io/531w16/notes08/notes8.html



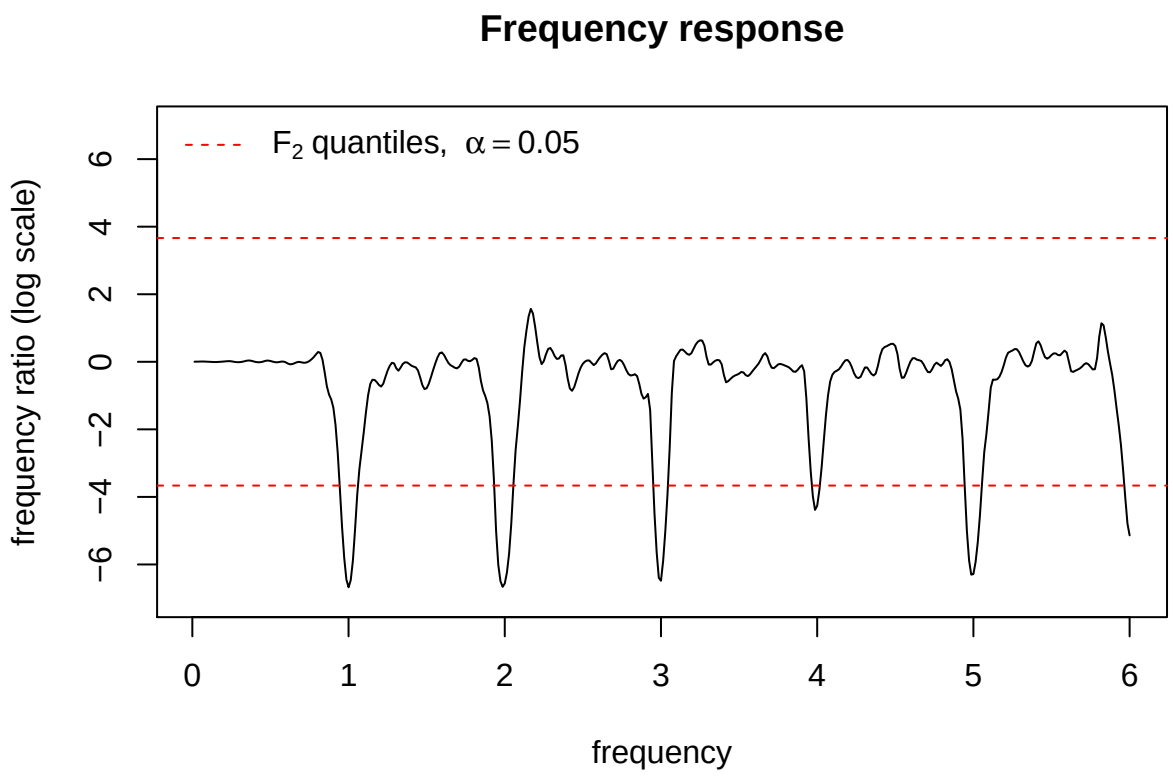
> We see seasonal variation, and perhaps we see business cycles on top of a slower trend. [...] The seasonal variation looks like an additive effect, say an annual fluctation with amplitude around 1 percentage point.

We are interested here in comparing the spectrum of the two series, so we won't differentiate the series. We can use the approximate Fisher $F_{2,2}$ distribution to perform a two-sided test for the null hypothesis of equality of spectra.

```
perio <- spec.pgram(fast = TRUE, ts.union(u1, u2), spans = c(3, 5, 3), taper = 0.2,
    plot = FALSE)
plot(perio, log = "dB", sub = "", main = "US unemployment.\n Raw (black) and seasonally adjusted (red)")
```



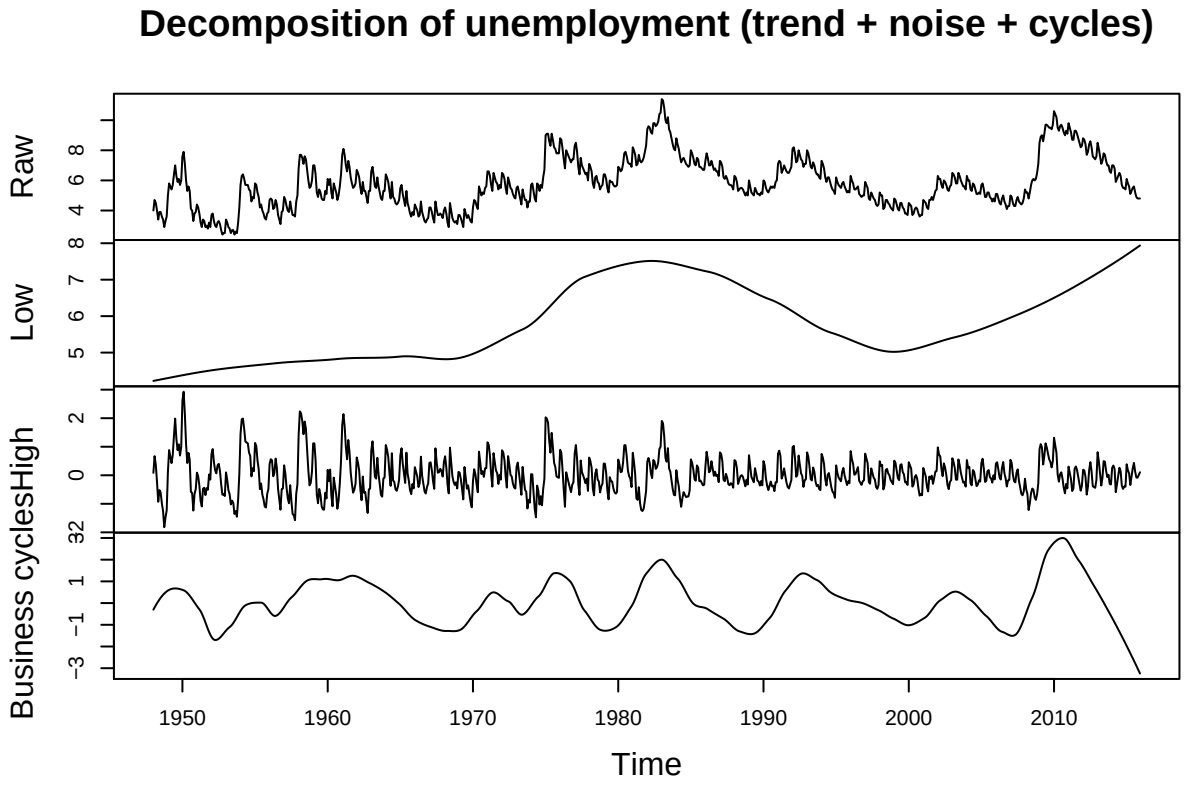

```
plot(perio$freq, log(perio$spec[, 2]/perio$spec[, 1]), type = "l", ylab = "frequency ratio (log scale)"
    xlab = "frequency", main = "Frequency response", ylim = c(-7, 7))
abline(h = log(c(qf(0.025, 2, 2), qf(0.975, 2, 2))), lty = "dashed", col = "red")
legend("topleft", lty = 2, col = 2, legend = expression(F[2, 2] ~ "quantiles, " ~
    alpha == 0.05), bty = "n")
```



What frequencies where removed by the seasonally adjusted series and what time period does these correspond to?

One could identify business cycles or focus and separate high frequency and low frequency by using low pass or high pass filters. This is basically smoothing. One could use loess, which uses local linear fit, to achieve this, but any other smoother discussed above could also achieve the objective. The argument `span` in `loess` is the bandwidth parameter (the larger, the more observations are taken into account).

```
# Low pass and high pass filter
u_low <- ts(loess(u1 ~ time(u1), span = 0.5)$fitted, start = 1948, frequency = 12)
u_hi <- ts(u1 - loess(u1 ~ time(u1), span = 0.1)$fitted, start = 1948, frequency = 12)
u_cycles <- u1 - u_hi - u_low
plot(ts.union(Raw = u1, Low = u_low, High = u_hi, `Business cycles` = u_cycles),
    ylab = "main", main = "Decomposition of unemployment (trend + noise + cycles)")
```



4.3.2 Exercise 1: Southern oscillation index (SOI) and fish recruitement

The data `soi` from the package `astsa` contains 453 monthly measures of the SOI. The dataset `rec` contains fish recruitement statistics for the same period.

1. Perform a spectral analysis using a parametric estimator based on an AR model.
2. Plot the log-periodograms of both series on the decibel scale
3. Identify the main frequencies of the series.
4. Try out different tapers and different degrees. Note the impact of the latter.
5. Smooth the series using a low-pass filter and extract the seasonal variation. Plot the periodogram and comment on the resulting estimate.

Chapter 5

Covariates and dynamic linear models

This tutorial addresses the following:

- estimation of ARIMA-GARCH models with parameter constraints
- inclusion of covariates in the specification of the variance
- model prediction based on simulations
- estimation of a dynamic linear model using the packages `dlm` and `KFAS`
- confidence intervals for quantile-quantile plots.

5.1 Simulation-based prediction intervals for ARIMA-GARCH models

In many cases, residuals from SARIMA models exhibit stochastic volatility (the variance is not constant). Since there is no function (to the best of my knowledge) to fit a SARIMA-GARCH model, you can do so in multiple steps. That is, first first your SARIMA and use the residuals from the latter and plug-them as data for your GARCH model. The point estimates will be consistent and asymptotically normal, but the standard errors are incorrect and need to be adjusted (one uses so-called **robust standard errors**). Since the confidence intervals from `arima` are normal confidence intervals, you will need to adjust them manually. The following recipe can help you get prediction intervals manually by means of simulation.

The h-step ahead forecasts are best linear predictors, meaning they are fixed. An alternative is to simulate trajectories from the fitted models conditional on the observed past and use the latter to obtain prediction intervals. It is crucial to take into account previous errors and observations so that our simulations make sense.

To generate K step ahead forecasts from an ARIMA-GARCH model:

1. Let $v_t = \varepsilon_t \sigma_t$ and $\varepsilon \sim F(0,1)$, where F is e.g. a Normal or a Student-t distribution. Generate GARCH innovations based on $\widehat{v}_{1:n}$ and $\widehat{\sigma}_{1:n}$, estimating σ^n_{n+1} using the GARCH recursions and sampling ε_t, form a new variable $\widetilde{v}_{n+1} = \sigma^n_{n+1}\varepsilon_{n+1}$. Iterate $K+d$ times, where d is the number of differences.
2. Pass to `arima.sim` the following arguments

- `model`: a list or the output of `arima` or variant thereof
- `n`: the integer $K+d$
- `innov`: the simulated series $\widetilde{v}_{n+1}, \ldots, \widetilde{v}_{n+K}$ as
- `start.innov`: the vector of residuals $\widehat{v}_{1:n}$.
- `n.start`: the integer $n-1$.

The function will thus return a time series drawn from your fitted ARIMA-GARCH model. Replicate this procedure $B = 1000$ times, say, then use as pointwise prediction intervals the 95% confidence interval based on the simulated values with rank 25 and 975. You could use the mean or median of the simulated trajectory as point forecast.

If you use `fGarch` package, the model adjusts for the distribution and for your ARMA-GARCH model; see the help in `?'predict,fGARCH-method'`.

If you wish to fit the ARIMA-GARCH jointly, but try to constrain some parameters to mimic a SARIMA model, this is also possible using the `rugarch` package by fixing some of values of $\boldsymbol{\theta}, \boldsymbol{\phi}$ to zero. Note that the optimization in `rugarch` will then often fail because the fit from the `arima` part is not stationary. Fixing parameters allows one to only estimate a handful of parameters.

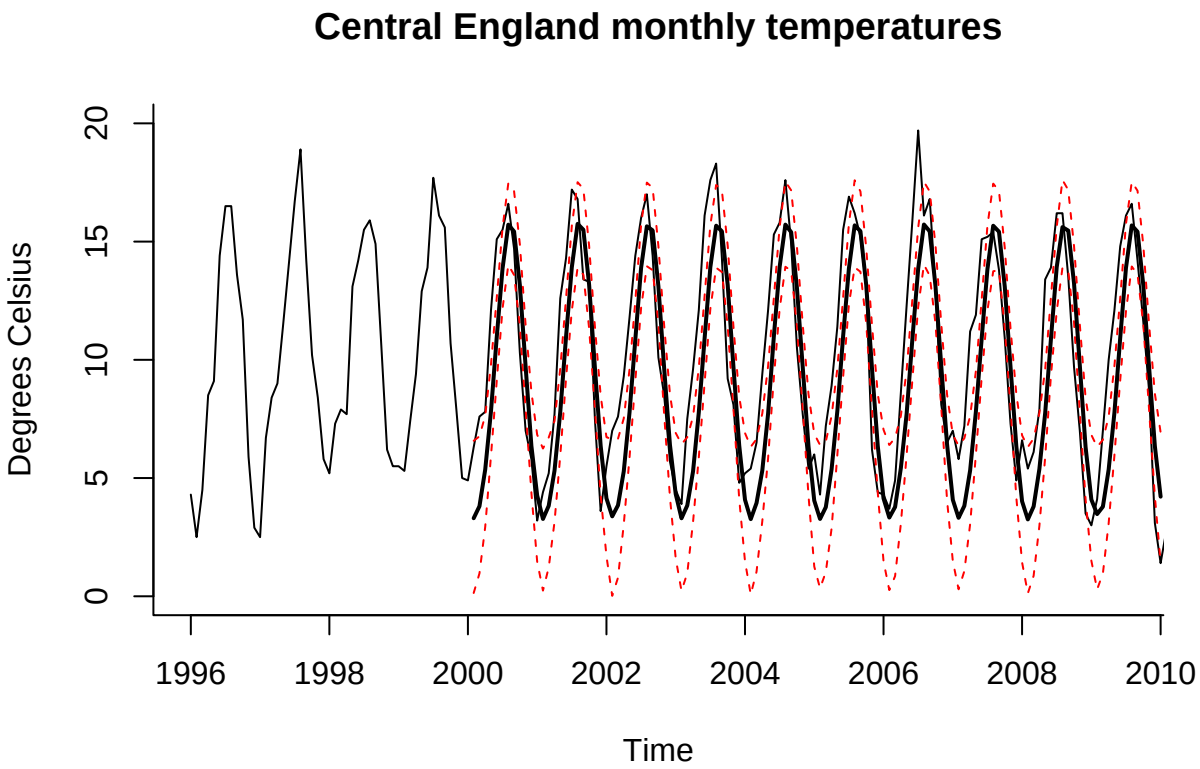
Here is an example using monthly temperatures. Fitting an ARIMA without constraints with the same specification would require estimating 28 additional ARMA parameters.

```
library(rugarch)
# Mean monthly temperature in UK, from Hadley center
data(CETmonthly, package = "multitaper")
# Keep period 1900-2000
mtempfull <- ts(CETmonthly[, 3], start = c(CETmonthly[1, 1], CETmonthly[1, 2]),
    frequency = 12)
mtemp <- window(mtempfull, start = c(1900, 1), end = c(1999, 12))
months <- rep(1:12, 100)
# Fit our favorite model - Fourier basis + SARMA(1,0,1)x(1,0,1)
meanreg <- cbind(cos(2 * pi * months/12), sin(2 * pi * months/12), cos(4 * pi *
    months/12), sin(4 * pi * months/12))
sarima <- forecast::Arima(mtemp, order = c(3, 0, 1), seasonal = c(1, 0, 1),
    xreg = meanreg)
# Not great, but will serve for the illustration

# This SARIMA is basically an ARMA model of high order with constraints
# These constraints can be roughly replicated by fixing most components to
# zero For example, if we have a SARMA (0,1)x(0,1)[12], coefficients 2 to 11
# are zero We will however estimate an extra MA parameter at lag 13 which
# would normally be prod(sarima$model$theta[c(1,12)])

# Extract AR coefficients via $model$phi Extract MA coefficients via
# $model$theta Find which parameters are zero and constrain them
lnames <- c(paste0("ar", which(sapply(sarima$model$phi, function(th) {
    isTRUE(all.equal(th, 0))
}))), paste0("ma", which(sapply(sarima$model$theta, function(th) {
    isTRUE(all.equal(th, 0))
}))))
constraints <- rep(list(0), length(lnames))
names(constraints) <- lnames
order <- c(length(sarima$model$phi), length(sarima$model$theta))

# To have a non-constant variance, we can let it vary per month Could try
# with dummies, but will stick with Fourier (issues with convergence of
# optimization routine otherwise) Creating a matrix of dummies varreg <-
# model.matrix(lm(months~as.factor(months)))[,-1] Model specification
model <- ugarchspec(variance.model = list(model = "eGARCH", garchOrder = c(0,
    0), external.regressors = meanreg), mean.model = list(armaOrder = order,
    include.mean = TRUE, external.regressors = meanreg), distribution.model = "std",
    fixed.pars = constraints)
# Fit model
fitmodel <- ugarchfit(spec = model, data = mtemp)
# Coefficients of the model
```

```
fitmodel@fit$coef[fitmodel@fit$coef != 0]
```

```
          mu          ar1          ar2          ar3         ar12
 9.133351478  0.810482688 -0.082144383 -0.006738789 -0.403624702
        ar13         ar14         ar15          ma1         ma12
 0.903950549 -0.179310483 -0.034706531 -0.608517750  0.429732865
        ma13       mxreg1       mxreg2       mxreg3       mxreg4
-0.804101668 -4.870149150 -3.902274279 -0.111705243  0.515741055
       omega       vxreg1       vxreg2       vxreg3       vxreg4
 0.439684060  0.442367489  0.298552838  0.110085160  0.236309381
       shape
65.572523216
```

```
# Not parsimonious, but all coefficients are significant

# Forecast K observations ahead
plotforecasted <- function(m = 60) {
    # Here, the variance depends only on the monthly index - deterministic
    # variation
    sig <- sqrt(exp(fitmodel@fit$coef["omega"] + meanreg[1:m, ] %*% coef(fitmodel)[startsWith(prefix =
        names(coef(fitmodel)))]))
    arma_par <- coef(fitmodel)[as.logical(startsWith("ar", x = names(fitmodel@fit$coef)) +
        startsWith("ma", x = names(fitmodel@fit$coef)))]
    # Generate replications from the ARIMA model and add the deterministic part
    simu_paths <- replicate(c(arima.sim(model = as.list(arma_par), n = m, innov = sig *
        rt(m, df = fitmodel@fit$coef["shape"]), start.innov = as.vector(residuals(fitmodel)),
        n.start = length(residuals(fitmodel)) - 1) + meanreg[1:m, ] %*% coef(fitmodel)[startsWith("mxreg
        x = names(fitmodel@fit$coef))] + coef(fitmodel)[1]), n = 1000)
    # Extract the sample quantiles from the Monte-Carlo replications of the
    # paths
    confinter <- apply(simu_paths, 1, function(x) {
        quantile(x, c(0.05, 0.5, 0.95))
    })
    # Plot the series with previous years
    plot(window(mtempfull, start = c(1996, 1)), xlim = c(1996, m/12 + 2000 -
        0.5), ylim = c(0, 20), main = "Central England monthly temperatures",
        ylab = "Degrees Celsius", bty = "l")
    # Add forecasts with 90% confidence intervals
    matplot(x = seq(2000 + 1/12, by = 1/12, length.out = m), y = t(confinter),
        lty = c(2, 1, 2), lwd = c(1, 2, 1), type = rep("l", 3), col = c(2, 1,
            2), add = TRUE)
}

# Plot the forecasts
plotforecasted(120)
```



In this specific example, since the variance is not autoregressive and depends only on covariates, we need not simulate paths conditional on the last observed values of $\widehat{v}_t$.

5.2 State-space models and the Kalman filter

The main packages for performing dynamic linear modelling are `dlm` and `KFAS` (another package, `dlmodeler`, unifies the interface between the two). The package `KFAS` has more functionalities — see the vignette and the examples in `?KFAS` for details).

The dataset under study, `AirPassengers`, contains a time series with the number (in thousands) of air passengers worlwide. We fit a local linear trend model with seasonality to the latter. This is the dynamic equivalent of a time series model with intercept, trend and monthly dummies. The local level component is equivalent to an ARIMA(0,1,1), so the trend component will likely be negligeable here if we allow the level to vary. It does change the forecasts, since doubly-differencing creates a linear trend in the forecast as we have seen before in the Mauna Loa CO_2 example.

The unknown components are the four variance coefficients (and tentatively the initial states). If the latter are not specified or estimated, they are taken to be diffuse, starting from zero. This mean we would need to discard the first smoothed state estimates.

The steps to be undertaken are when fitting a dynamic linear model

1. Specify the model structure, indicating which parameters are to be estimated. You can include deterministic components by fixing their respective variance to zero. For seasonal models with dummies, only the first variance should be non-zero.
2. Obtain maximum likelihood estimates numerically
3. Plug the estimated parameters in the model
4. Filter forward the estimates. If you did not estimated or specified the initial state `m0`, the latter will be zero and this will impact your filtered values
5. Perform smoothing-backwards on the filtered object

6. Obtain forecasts (this does not work in `dlm` with regression models, but can be done manually using the Kalman recursions).
7. Run the usual diagnostics on the residuals (filtered minus observed) to check the adequacy of your model

```
library(dlm)
data("AirPassengers")
#?"AirPassengers"
#Fit a local level with seasonal dummies
plot(log(AirPassengers), ylab = "log monthly total passengers (in thousands)",
     main = "Number of international air passengers")
#Keep some observations for forecast validation
y <- as.numeric(log(window(AirPassengers, end = c(1958))))
n <- length(y)
time <- as.vector(time(window(AirPassengers, end = c(1958))))
month <- as.factor(as.integer(time*12) %%12)
#Build the dynamic linear model  - specification with local level + seasonal component (dummies)
#Need to specify initial values for mean m0, otherwise forecasts are diffuse values from zero
#takes a few steps to adjust
build <- function(params) {
  level <- dlmModPoly(order = 2, dV = exp(params[1]), dW = c(exp(params[2:3])))
  seas <- dlmModSeas(frequency = 12, dV = exp(params[1]), dW = c(exp(params[4]), rep(0, 10)))  #stochas
  mod <- level + seas
  return(mod)
}
#Initial parameter values - four log-variance, mean and trend, seasonal effect
init <- rep(-5,4)
#Fit the DLM model - numerical optimization
fit <- dlmMLE(y = y, parm = init, build = build, method = "BFGS", control=list(trace = 10, maxit = 1000)
```

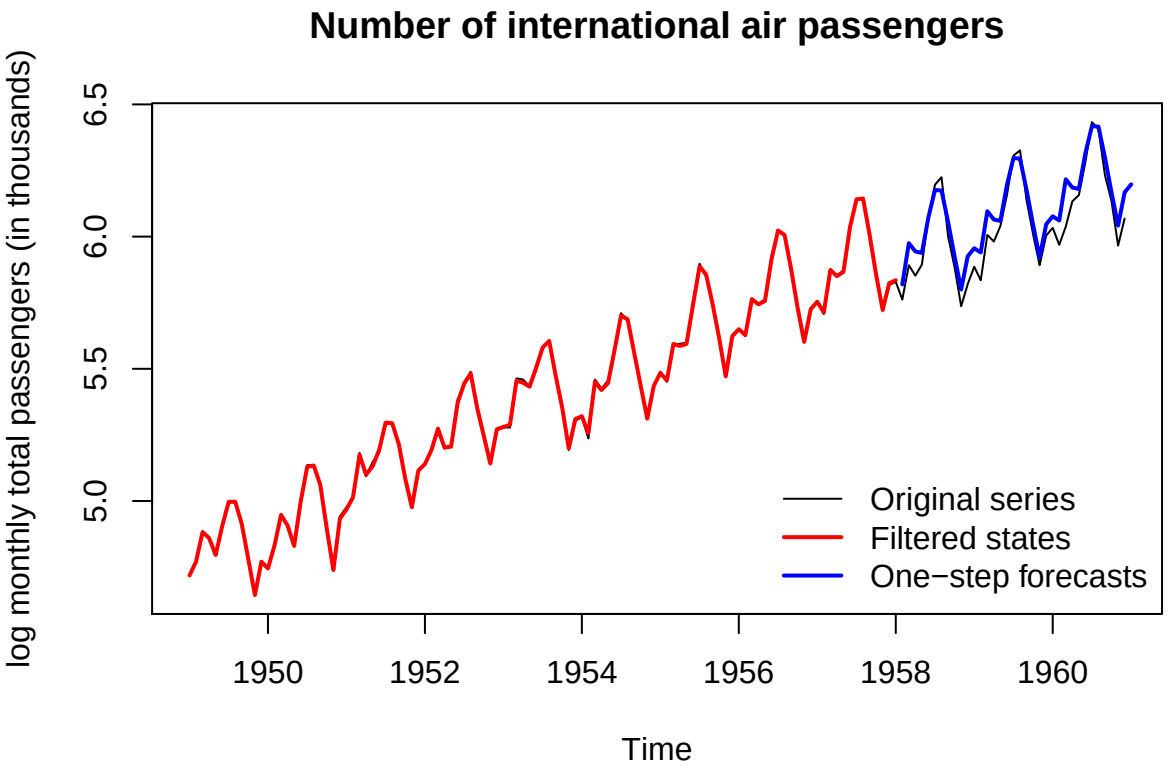
```
initial  value 8.105325
iter  10 value -151.637115
iter  20 value -151.688555
iter  30 value -151.689585
final  value -151.689701
converged
```

```
#Define model with estimated parameters
mod <- build(fit$par)

#Variance matrices
V(mod); #W(mod), but only diag(W(mod))[1:3] non-null
```

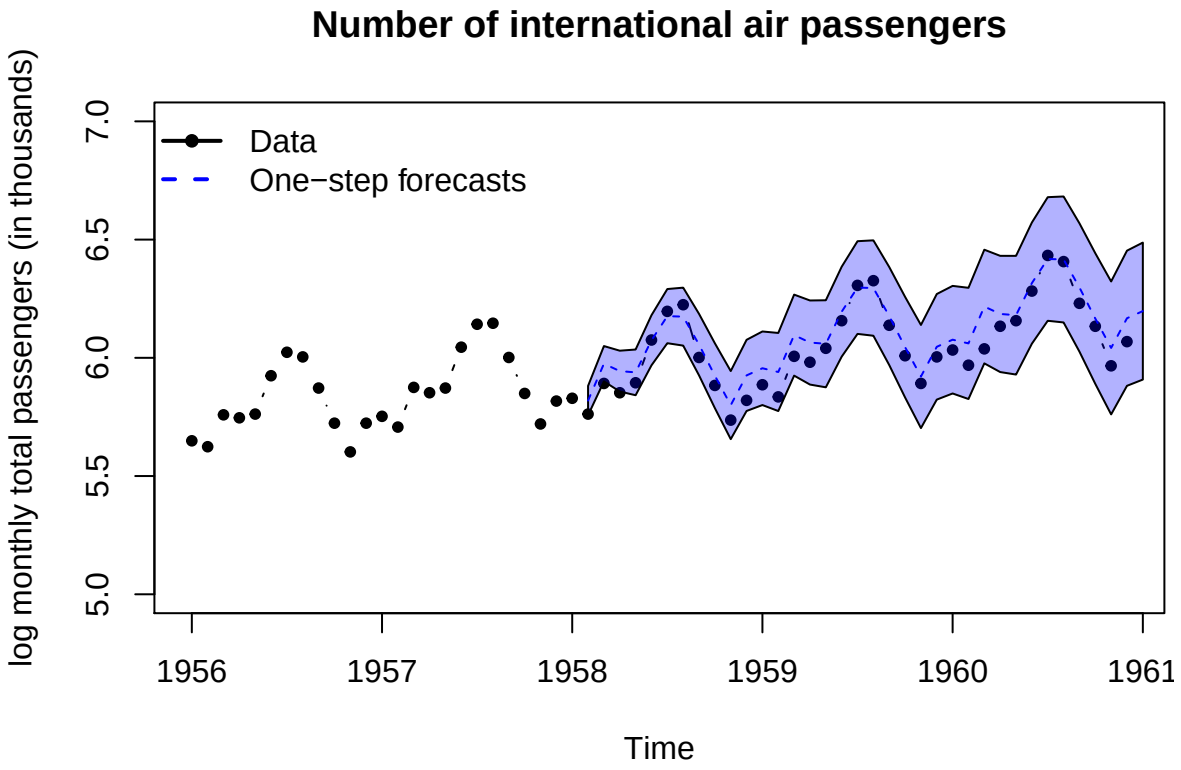
```
             [,1]
[1,] 0.0002514017
```

```
#We could remove the trend and mayhaps make the seasonal part deterministic

#Filtering
filtered <- dlmFilter(y, mod)
#plot(time, y, type = "l")
lines(time, c(mod$FF %*% t(filtered$m[-1,])), col = 2, lwd = 2) #filtered states

#One-step ahead forecasts (a linear fn of filtering mean)
```

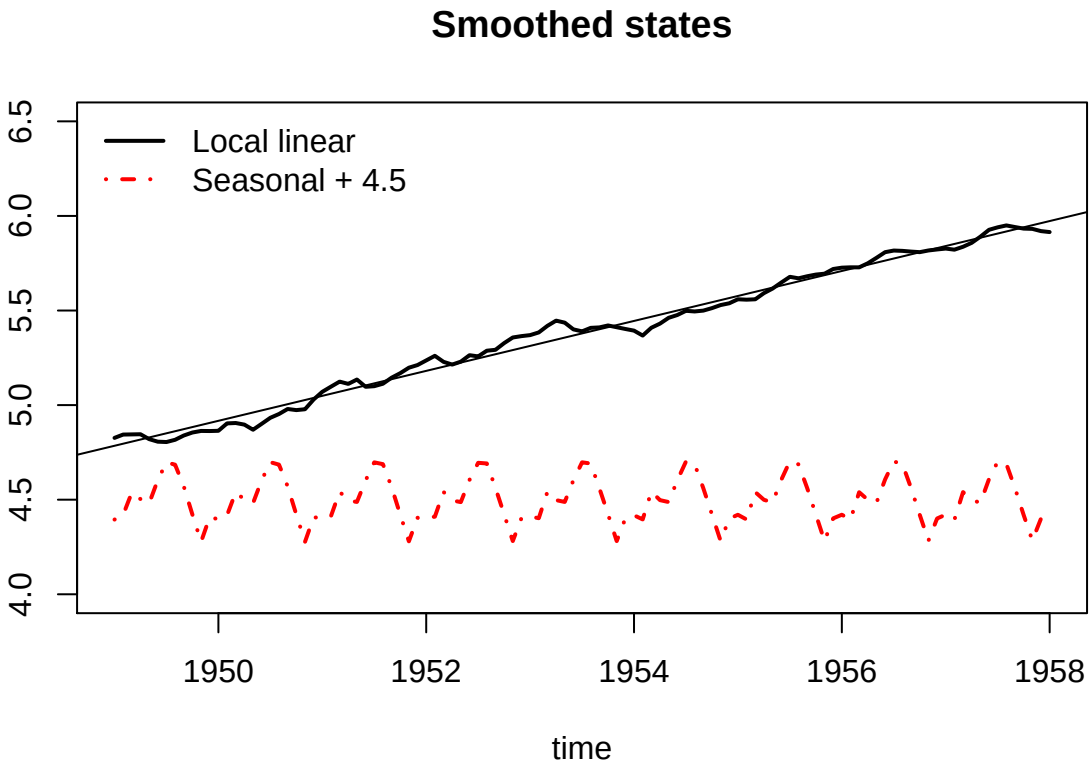
```
forecasted <- dlmForecast(filtered, nAhead = 36)
timelo <- seq(tail(time,1) + 1/12, by = 1/12, length = 36)
lines(timelo,  forecasted$f, col = 4, lwd=2)
legend("bottomright", c("Original series","Filtered states","One-step forecasts"),
       lty=c(1, 1, 1), lwd=c(1,2,2), col=c(1,2,4),bty = "n")
```




```
#90% Confidence intervals
plot(log(window(AirPassengers, start=1956)), ylab = "log monthly total passengers (in thousands)",
     main = "Number of international air passengers", ylim=c(5,7), type = "b", pch = 20)
polygon(x=c(timelo, rev(timelo)), y = c(forecasted$f + qnorm(0.95)*sqrt(unlist(forecasted$Q)),
        rev(forecasted$f - qnorm(0.95)*sqrt(unlist(forecasted$Q)))), col=scales::alpha("blue", alpha=0.
lines(timelo, forecasted$f, col = 4, lty = 2)
legend("topleft", c("Data","One-step forecasts"), lty=c(1, 2), lwd=2, col=c(1,4), pch=c(20,NA), bty = "
```




```
smoothed <- dlmSmooth(filtered)
plot(time, smoothed$s[-1,1], type= "l", ylab = "", ylim=c(4,6.5), lwd=2, main = "Smoothed states")
lines(time, 4.5 + smoothed$s[-1,3], col = 2, lwd = 2, lty = 4)
legend("topleft", c("Local linear","Seasonal + 4.5"), lty = c(1, 4), lwd = 2, col = 1:2, bty = "n")
abline(lm(y ~ time)$coefficients)
```



Let us do the same analysis, this time with `KFAS`. The model is defined analogously, but this time, the parameters to be estimated are left as `NA`.

```
y2 <- log(window(AirPassengers, end = c(1958)))
library(KFAS)
model <- SSModel(y2 ~ SSMtrend(2, Q = list(matrix(NA), matrix(NA))) + SSMseasonal(period = 12,
    sea.type = "dummy", Q = NA), H = matrix(NA))
fit2 <- fitSSM(model, inits = c(0, 0, 0, 0), method = "BFGS")
# Maximum likelihood estimate for sigmas
exp(fit2$optim.out$par)
```

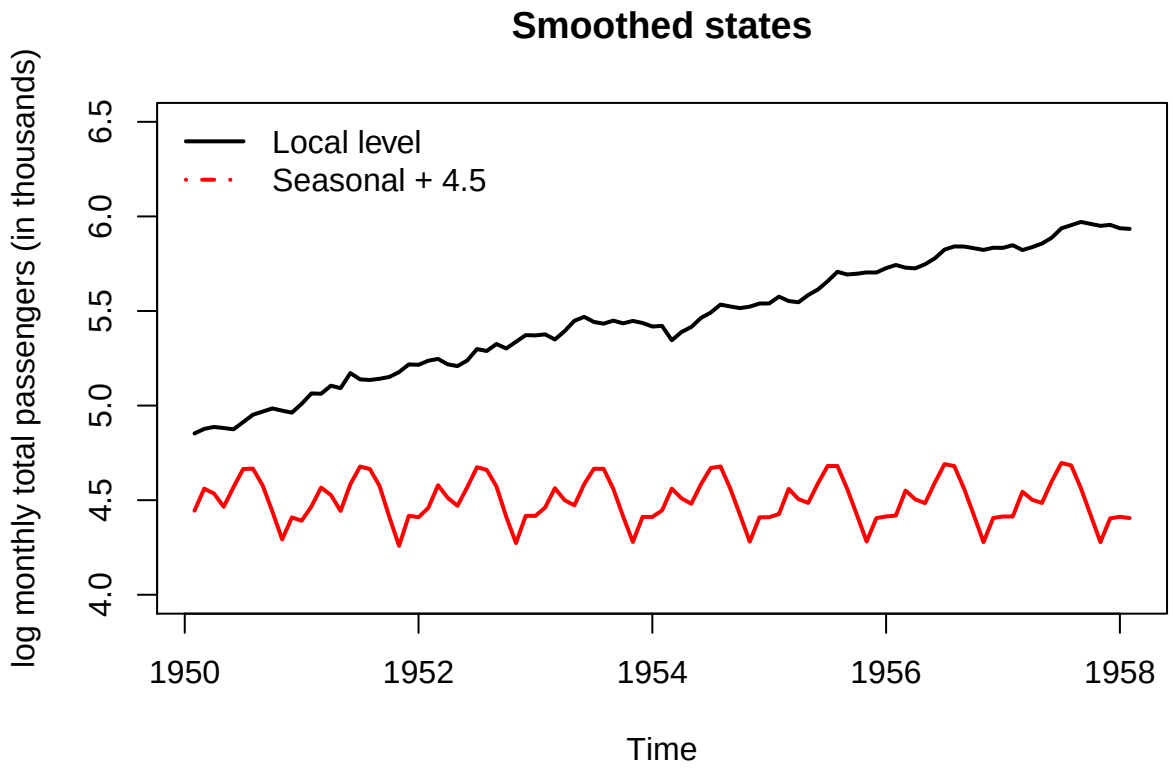
```
[1] 5.961276e-04 3.468949e-09 1.377059e-08 3.703626e-04
```

```
# Kalman filtering and smoothing
out <- KFS(fit2$model)
out
```

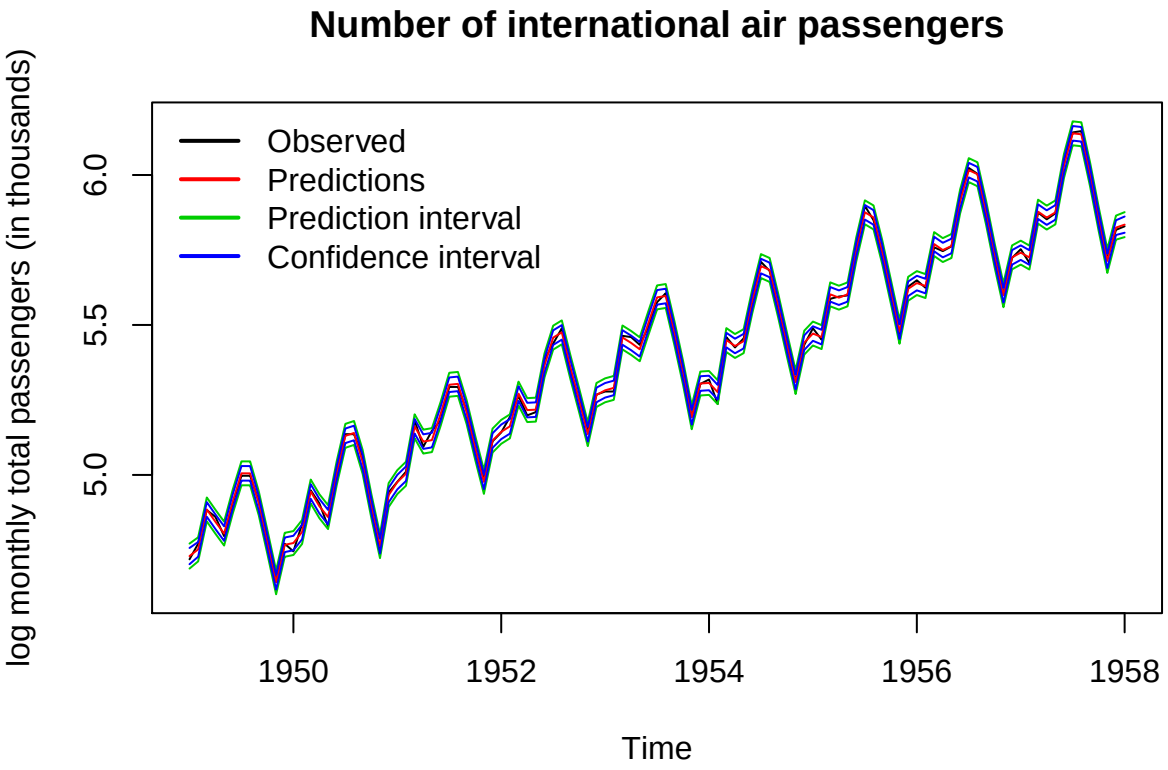
```
Smoothed values of states and standard errors at time n = 109:
             Estimate   Std. Error
level         5.923737   0.018009
slope         0.010225   0.002386
sea_dummy1   -0.089207   0.010058
sea_dummy2   -0.097948   0.010232
sea_dummy3   -0.219836   0.010212
sea_dummy4   -0.079318   0.010196
sea_dummy5    0.063466   0.010185
sea_dummy6    0.187795   0.010178
sea_dummy7    0.199432   0.010176
sea_dummy8    0.104437   0.010178
```

```
sea_dummy9   -0.014626   0.010185
sea_dummy10  -0.000807   0.010196
sea_dummy11   0.041156   0.010212
```

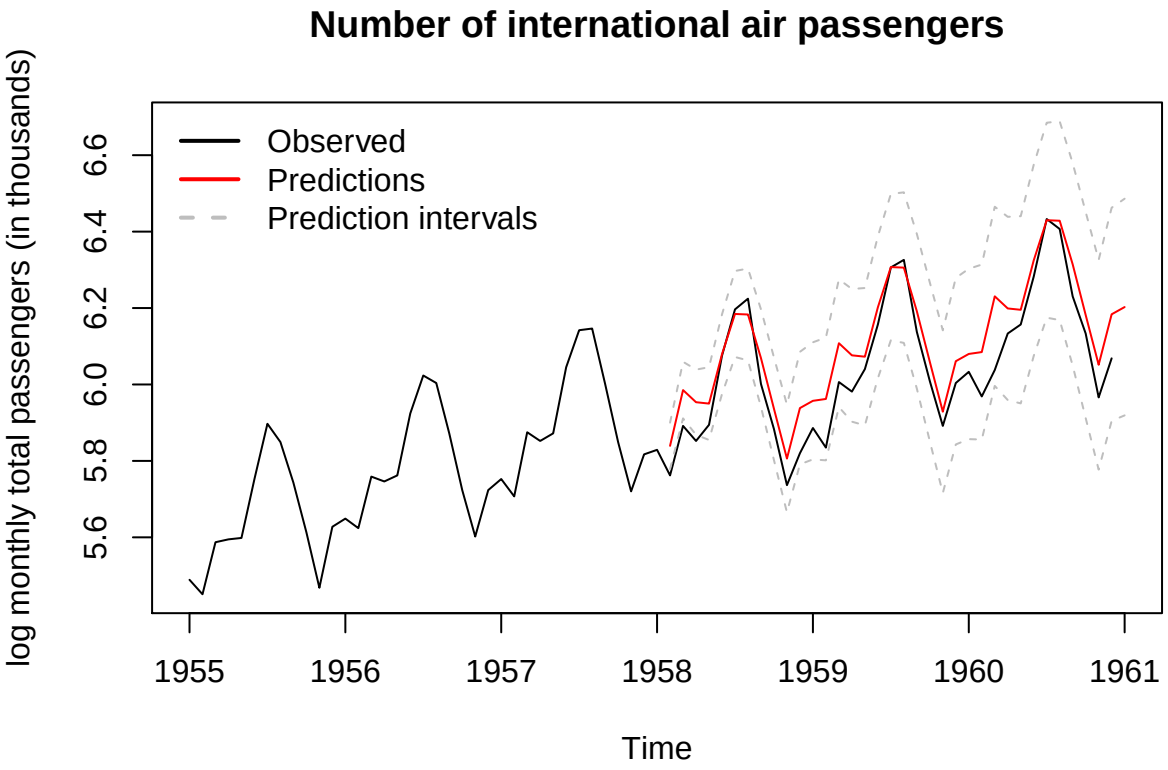
```
model_air <- fit2$model
# Confidence and prediction intervals for the expected value and the
# observations.  Note that predict uses original model object (with
# estimated parameters), not the output from KFS.
conf_air <- predict(model_air, interval = "confidence", level = 0.9)
pred_air <- predict(model_air, interval = "prediction", level = 0.9)

# Plot of estimated states If we do not estimate the initial states, we
# should get rid of first 12+1 first values Because they are used for the
# initialization
plot(window(out$a[, "level"], start = c(1950, 2)), type = "l", ylim = c(4, 6.5),
    ylab = "log monthly total passengers (in thousands)", lwd = 2, main = "Smoothed states")
# lines(window(out$a[,'slope'] + 5, start = c(1950, 2)), col = 3, lwd = 2)
lines(window(out$a[, "sea_dummy1"] + 4.5, start = c(1950, 2)), col = 2, lwd = 2)
legend("topleft", c("Local level", "Seasonal + 4.5"), lty = c(1, 4), lwd = 2,
    col = 1:2, bty = "n")
```




```
# Filtering backwards plus one-step ahead forecasts Plot has one-step ahead
# forecasts, confidence intervals and one-step ahead prediction intervals
ts.plot(cbind(y2, pred_air, conf_air[, -1]), col = c(1, 2, 3, 3, 4, 4), ylab = "log monthly total passe
    main = "Number of international air passengers")
legend("topleft", c("Observed", "Predictions", "Prediction interval", "Confidence interval"),
    lty = 1, lwd = 2, col = 1:4, bty = "n")
```




```
# Predictions ahead
pred_air <- predict(model_air, interval = "prediction", level = 0.9, n.ahead = 36)
ts.plot(cbind(log(window(AirPassengers, start = 1955, end = 1961)), pred_air),
    col = c(1, 2, "grey", "grey"), ylab = "log monthly total passengers (in thousands)",
    main = "Number of international air passengers", lty = c(1, 1, 2, 2))
legend("topleft", c("Observed", "Predictions", "Prediction intervals"), lty = c(1,
    1, 2), lwd = 2, col = c(1, 2, "grey"), bty = "n")
```



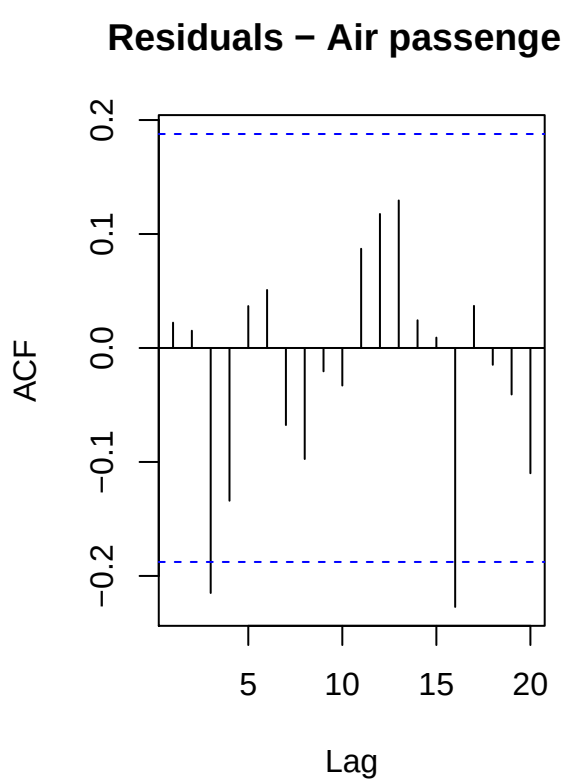
It now remains to perform some validation diagnostics. We check first the (partial) correlograms to ensure that there is no residual autocorrelation, then test the normality of the residuals and the hypothesis that they are white noise.

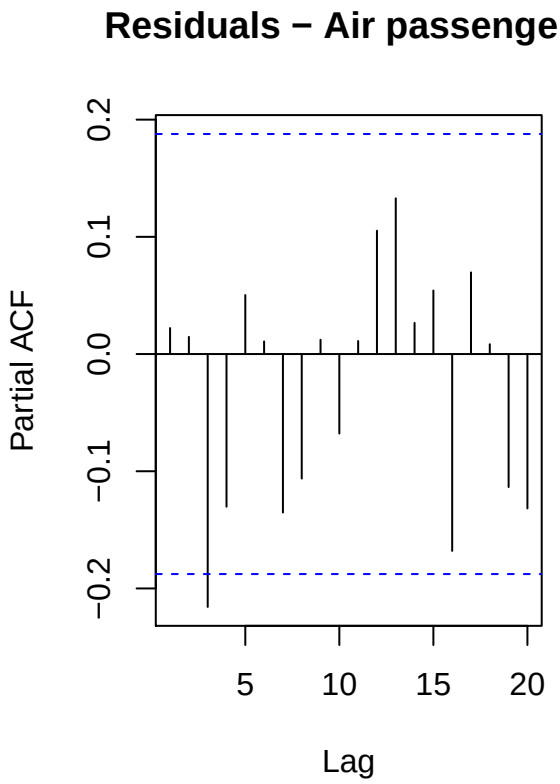
To add a confidence interval to the QQ-plots, we can use two methods. Recall that the standard normal quantile-quantile plot has - on the x axis, the theoretical quantiles, $\Phi^{-1}(\text{rank}(X_i)/(n+1))$ - on the y axis, the empirical quantiles, X_i

Under the null hypothesis that the observations are independent and identically distributed from $\mathcal{N}(0,1)$

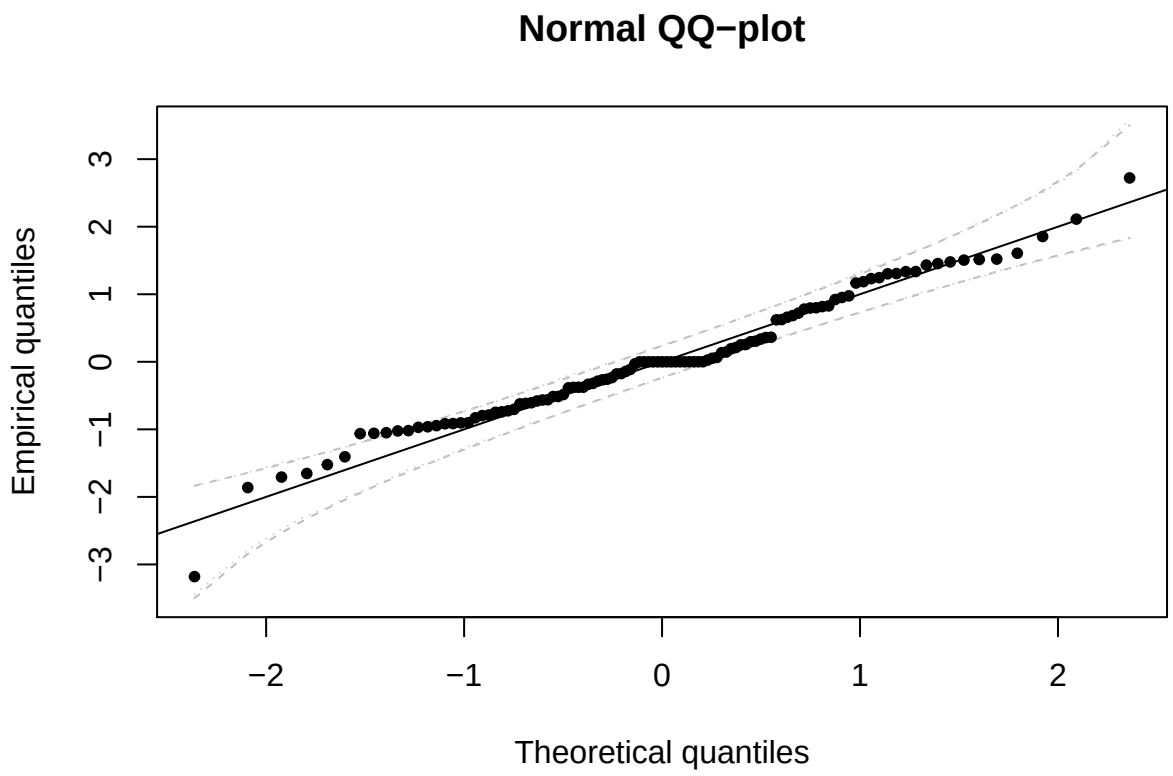
- the ith order statistic of the uniform $U_{(i)} = \Phi^{-1}(X_{(i)})$ is distributed as Beta $\mathcal{B}(i, n-i+1)$. Thus, a 95% pointwise confidence interval can be obtained by taking the 2.5 and 97.5 percentiles of the Beta and applying the transformation Φ^{-1} to the latter.
- We can resort to the bootstrap and simulate $B = 1000$ samples under the null hypothesis. Here, we would thus generate B samples from $\mathcal{N}(0,1)$ of size n and order each of them. This yields $\{X_{i,b}\}$ for $i = 1,\ldots,n$ and $b = 1,\ldots,B$ where $X_{1,b} < X_{2,b} < \ldots, X_{n,b}$. It remains to take the 2.5 and 97.5 percentiles of the ith order statistic $\{X_{i,b}\}, b = 1,\ldots,B$. Note that if your residuals were $\mathcal{N}(\mu,\sigma^2)$ rather than $\mathcal{N}(0,1)$ and you scaled them by subtracting $\hat{\mu}$ and dividing by $\hat{\sigma}$, you should scale the replicate samples in an analogous fashion by estimating the mean and variance of each of the B samples.

```
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(1, 2))
TSA::acf(resid(filtered)$res, main = "Residuals - Air passengers")
pacf(resid(filtered)$res, main = "Residuals - Air passengers")
```

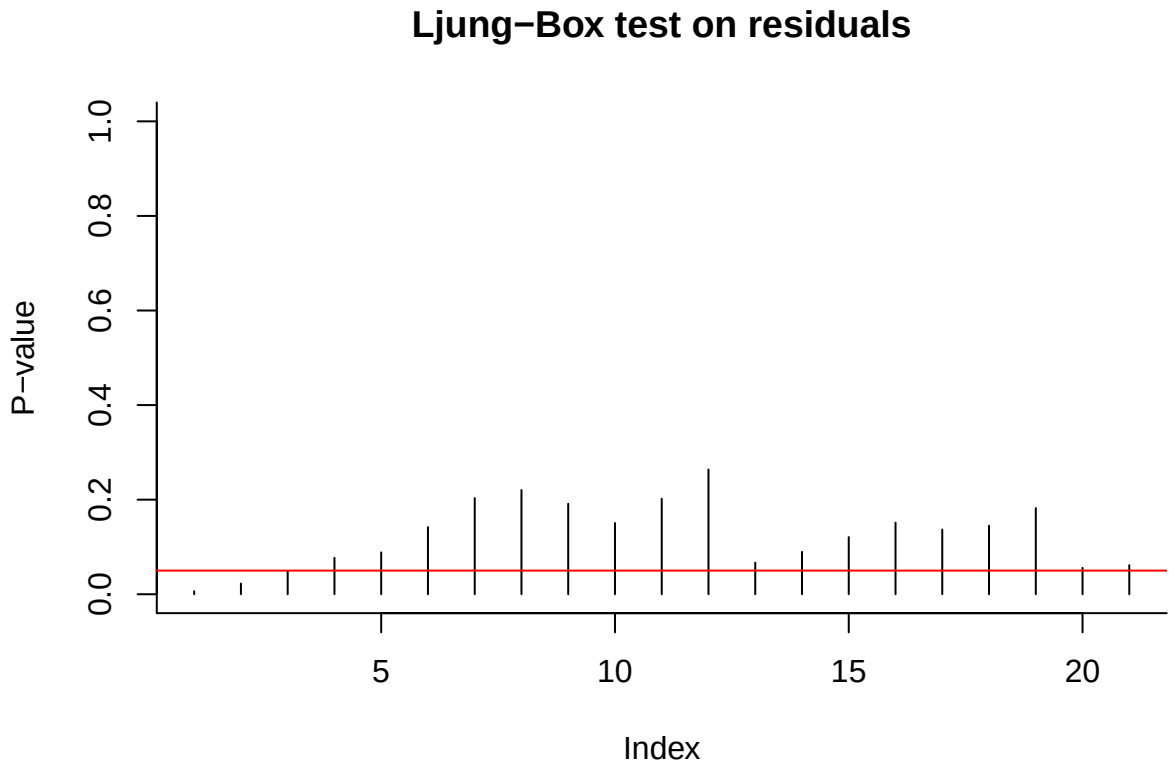





```r
par(oldpar)

# Normal quantile-quantile plot with confidence intervals
res <- resid(filtered)$res
n <- length(res)
quant <- qnorm(rank(res)/(n + 1))
# Pointwise limits based on distribution of order statistics
confint_lim <- t(sapply(1:n, function(i) {
    qnorm(qbeta(c(0.025, 0.975), i, n - i + 1))
}))
matplot(sort(quant), confint_lim, type = "l", lty = 2, col = "grey", main = "Normal QQ-plot",
    xlab = "Theoretical quantiles", ylab = "Empirical quantiles")
# Pointwise limits based on draws from null model
confint_lim2 <- t(apply(apply(matrix(rnorm(1000 * n), nrow = n, ncol = 1000),
    2, sort), 1, function(x) {
    quantile(x, c(0.025, 0.975))
}))
matplot(sort(quant), confint_lim, type = "l", lty = 2, col = "grey", main = "Normal QQ-plot",
    xlab = "Theoretical quantiles", ylab = "Empirical quantiles")
matplot(sort(quant), confint_lim2, type = "l", lty = 3, col = "lightgrey", add = TRUE)
points(quant, res, pch = 20)
abline(c(0, 1))
```




```r
# Ljung-Box test on residuals
plot(sapply(4:24, function(l) {
    Box.test(x = res, lag = l, type = "L", fitdf = 3)$p.value
}), type = "h", ylim = c(0, 1), ylab = "P-value", main = "Ljung-Box test on residuals",
    bty = "l")
abline(h = 0.05, col = 2)
```



The diagnostic plots here indicate that the results are not quite white noise, but the correlogram indicates that there is no residual linear dependence (they are uncorrelated). The quantile-quantile plot shows that the fit is satisfactory - more than 95% of the values falls within the confidence bands.

5.2.1 Exercise 1: Dynamic linear model for the Nile river dataset

1. Try removing the trend component. Does your conclusions change?
2. Fix the variance component corresponding to the seasonal part to zero (making the effect deterministic). Test whether the latter was significant.
3. Remove the initial states from the analysis and compare the filtered estimates.
4. Repeat the analysis on the `nile` dataset.
5. Add an intervention covariate to the model for `nile`, corresponding to a structural break in level in 1902 due to the Assouan dam construction.

Chapter 6

Notes on irregular time series and missing values"

6.1 Irregular time series

The most common class for time series is `ts`, but I prefer to work with `xts` since the latter handles irregular time series and has subsetting options not available for `ts` objects. You can convert strings (or numerical) to `Date` using `as.Date`. See `?DateTimeClasses` for more information about the classes used to represent date and time objects.

```
library(xts)
library(lubridate)
```

6.1.1 Exercise 1: Jussy air temperature

1. Load the data from the website, available at http://sma.epfl.ch/~lbelzile/math342/jussy_temp.csv, using the command

```
jussy <- read.csv("http://sma.epfl.ch/~lbelzile/math342/jussy_temp.csv", header = TRUE,
    sep = ";", na.strings = "######", stringsAsFactors = FALSE)[, 1:2]
```

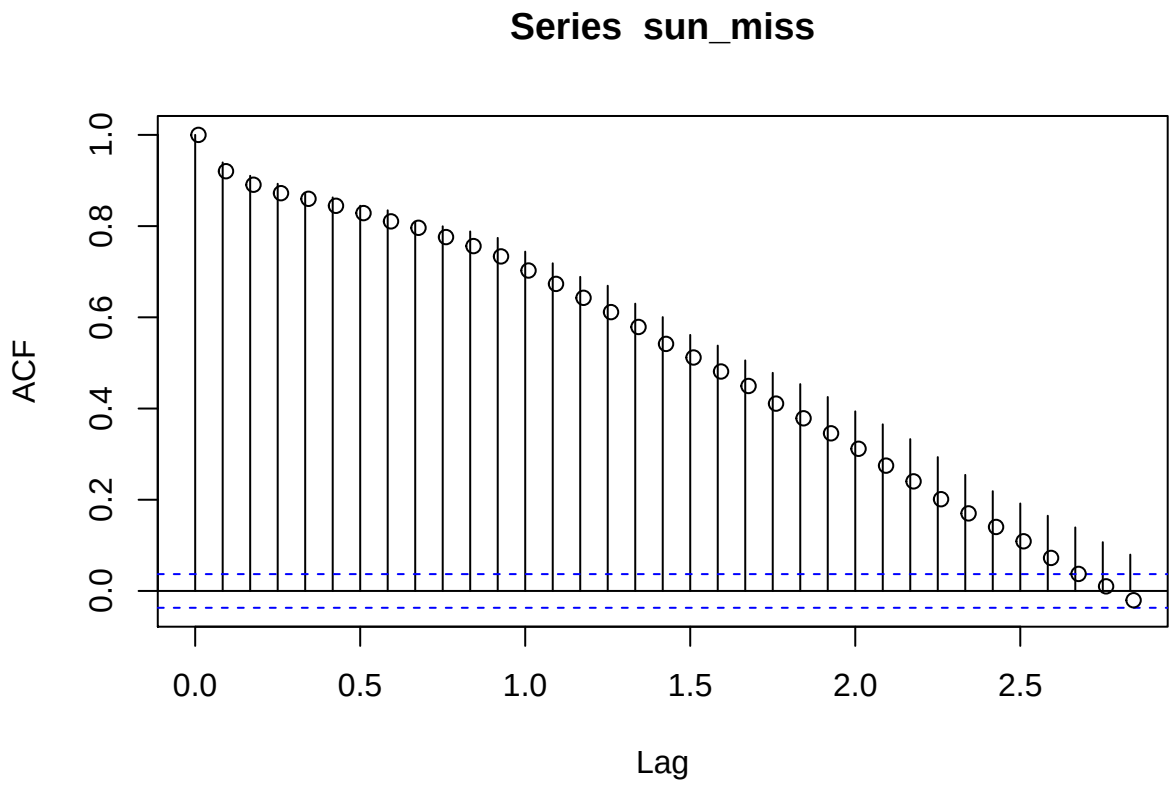
2. Extract the time object and mutate the object to keep only the daily maximum (`xts::apply.daily`).
3. Plot the daily maximum and comment on any particularity of the dataset (trend, seasonality, structural breaks, time window).
4. Find how many days on average per year have temperature below zero.
5. How many heatwaves have taken place during the data collection and how long did they last? For simplicity, define a heatwave as a period of more than 3 consecutive days during which the temperature exceeds 30C during the day, but does no down below 18C at night.

6.2 Imputation of missing values

If there are missing values in a time series, one ought to handle them with caution. This section adresses this problem, and is there mostly for students whose project dataset features missing values. A simple call to `summary` will tell you if your object contains `NA`s.

Suppose we remove some values from `sunspots`.

```
sun_miss <- sunspots
# We remove 300 values at random
sun_miss[ints <- sample.int(length(sunspots), 300, replace = FALSE)] <- NA
# The acf function will return an error if we do nothing acf(sun_miss) Ask
# the function to omit those values from the calculations (correct way if we
# keep them)
correlo_pass <- acf(sun_miss, na.action = na.pass)
correlo_excl <- acf(sun_miss, na.action = na.exclude, plot = FALSE)  #equivalent acf(na.omit(as.vector(
# The second is incorrect, because it changes the labels
points(correlo_pass$lag + 0.01, correlo_excl$acf)
```



The output is slightly different, but moreover the time stamps are off! This loss of information is even more dramatic if there are multiple consecutive values missing, which may distort the seasonality. Some datasets, for example financial time series, are **irregular**. This is due to closure of the stock market on holydays and week-ends, so there are apriori no missing values there. Just work with the classes `zoo` or `xts` and use `na.pass` as argument. This way, however, you won't ever compute lag one correlation between Friday and Monday, but will classify the empirical estimates as three days. You can also remove the weekly seasonality first and later use `na.remove` (just be careful with your interpretation then).

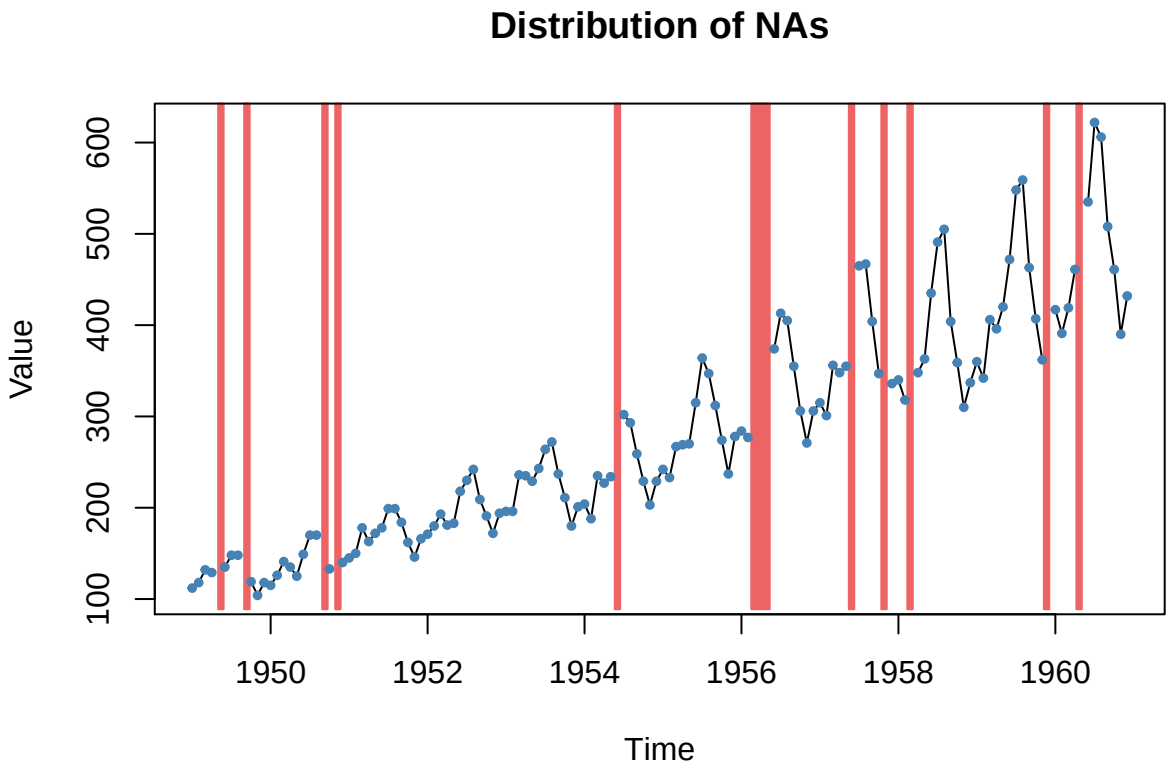
If your series has values that are missing at random and there is very few of them (1%), you could as a preliminary step impute them. Including new datapoints will bias your standard errors (you are adding information that was not present in the original dataset) unless you adjust for this carefully.

##Diagnostics for missing values and smoothing

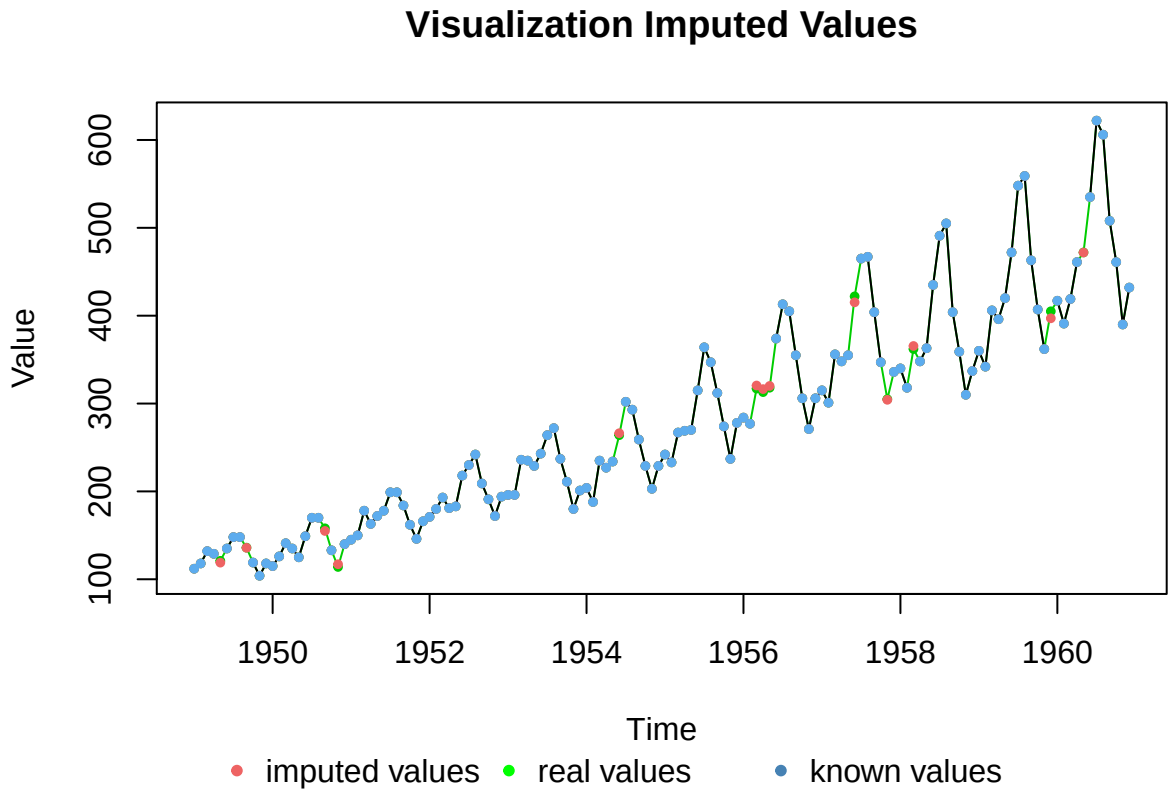
The package `stlplus` handles missing values, contrary to `stl`. Likewise, there are utilities in `zoo` to perform linear interpolation or use smoothing estimates from a seasonal Kalman filter (which we will cover at the end of the course). These are available respectively under the name `zoo::na.approx` and `zoo::na.StrucTS` (see also

the help file). More sophisticated methods can be found in the package `imputeTS`. The latter provides more tools for plotting data with missing values (`plotNA.distribution`) and obtaining summary statistics out of the box (`statsNA`).

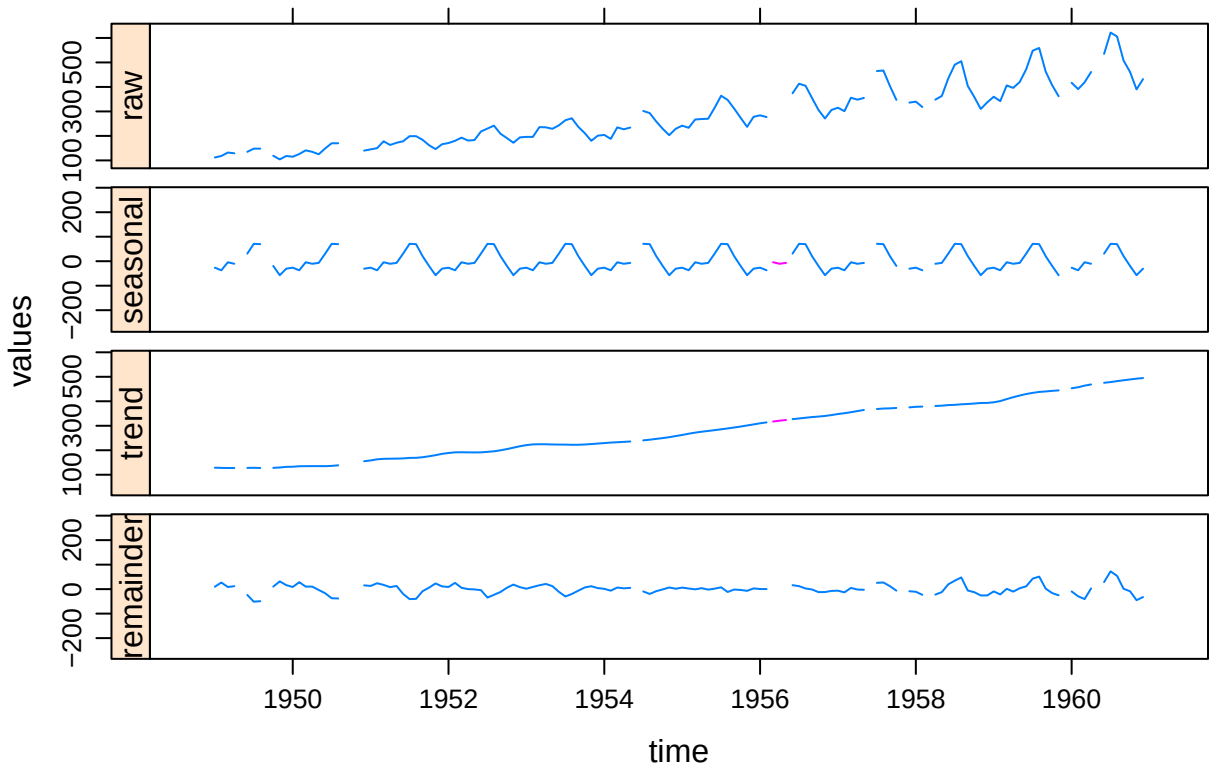
```
library(imputeTS)
plotNA.distribution(tsAirgap)
```




```
plotNA.imputations(x.withNA = tsAirgap, x.withImputations = na.seadec(tsAirgap,
    "kalman"), x.withTruth = tsAirgapComplete)
```




```
# Install package if not already present, otherwise load it
if (suppressWarnings(!require(stlplus))) {
    install.packages("stlplus")
    library(stlplus)
}

plot(stl_Airgap <- stlplus(tsAirgap, s.window = "periodic"))
```




```
# Increasing variance with number of air passengers - would need to
# transform the series to stabilize the variance
```

The following illustrates the use of local fit to interpolate the missing values, but one could equally well fit using a local linear model with `loess` and use the fitted values by predicting at unobserved time points. In general, these predictions are *wrong* because they do not include any time dependence structure. Another useful feature from the package `zoo` is `na.trim` to removing trailing `NA`s at the beginning and the end of a series.

6.2.1 Exercise 2: Tyne river flow

1. Import the following dataset and look at the summary

```
tyne <- read.csv(file = "http://sma.epfl.ch/~lbelzile/math342/23001-Tyne_at_Bywell.csv",
    header = FALSE, sep = ",", skip = 16, col.names = c("time", "height", "flag"),
    as.is = TRUE, na.strings = "NA")[, 1:2]
```

2. The dataset contains missing values. Transform `tyne` into an object of class `ts`. Plot the series with `plotNA.distribution`. Comment on the implications of imputing those values and on the values of the (partial) correlogram.
3. Try using `zoo::na.locf`. What does the function do?
4. Perform an `stl` decomposition with `stlplus` and comment on the output.

6.3 Solutions to Exercises

6.3.1 Exercise 1: Jussy air temperature

```
jussy <- read.csv("http://sma.epfl.ch/~lbelzile/math342/jussy_temp.csv", header = TRUE,
    sep = ";", na.strings = "######", stringsAsFactors = FALSE)[, 1:2]
library(xts)
library(lubridate)
colnames(jussy) <- c("time", "temp")
jussy_xts <- with(jussy, xts(temp, order.by = ymd_h(time)))
jussy_max <- apply.daily(jussy_xts, max)
number_freeze <- apply.yearly(jussy_xts, function(serie) {
    sum(I(apply.daily(serie, min) < 0), na.rm = TRUE)
})
# Exclude last incomplete year
mean(number_freeze[-length(number_freeze)])
```

```
[1] 85.53846
```

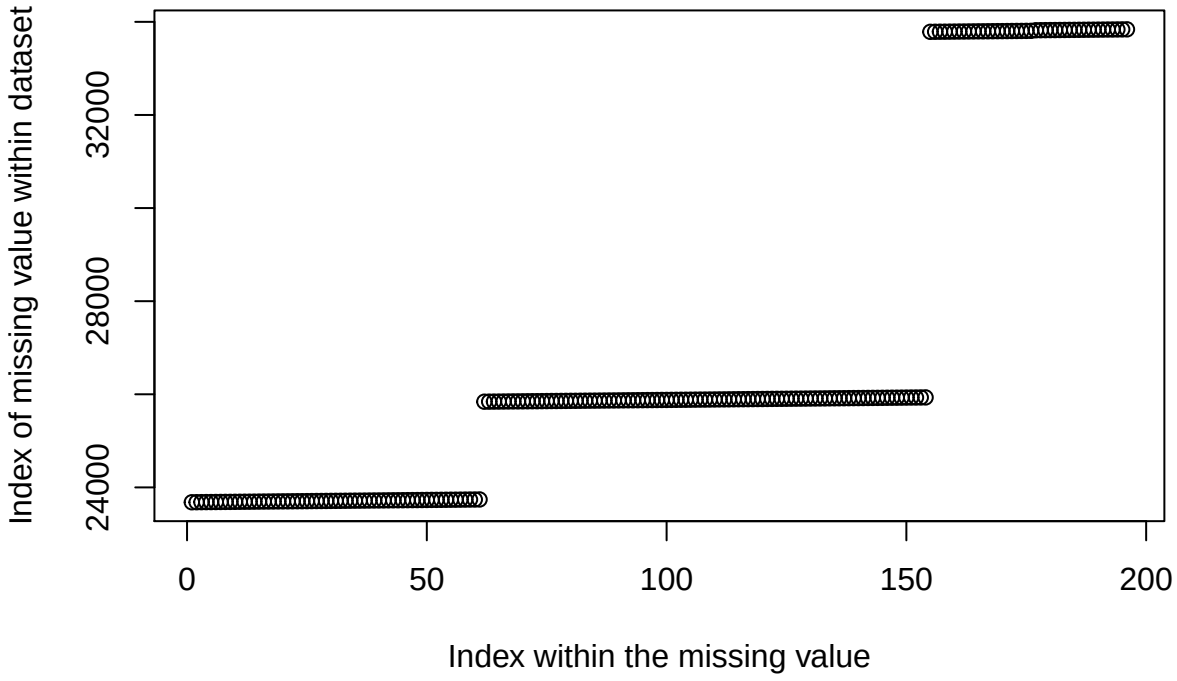
```
heatwave_ind <- intersect(which(apply.daily(jussy_xts, min) > 18), which(apply.daily(jussy_xts,
    max) > 30))  #which values match the two constraints
y <- rle(diff(heatwave_ind))  #run length
sum(I(y$lengths[y$values == 1] > 3))  #how many are above 3 days
```

```
[1] 2
```

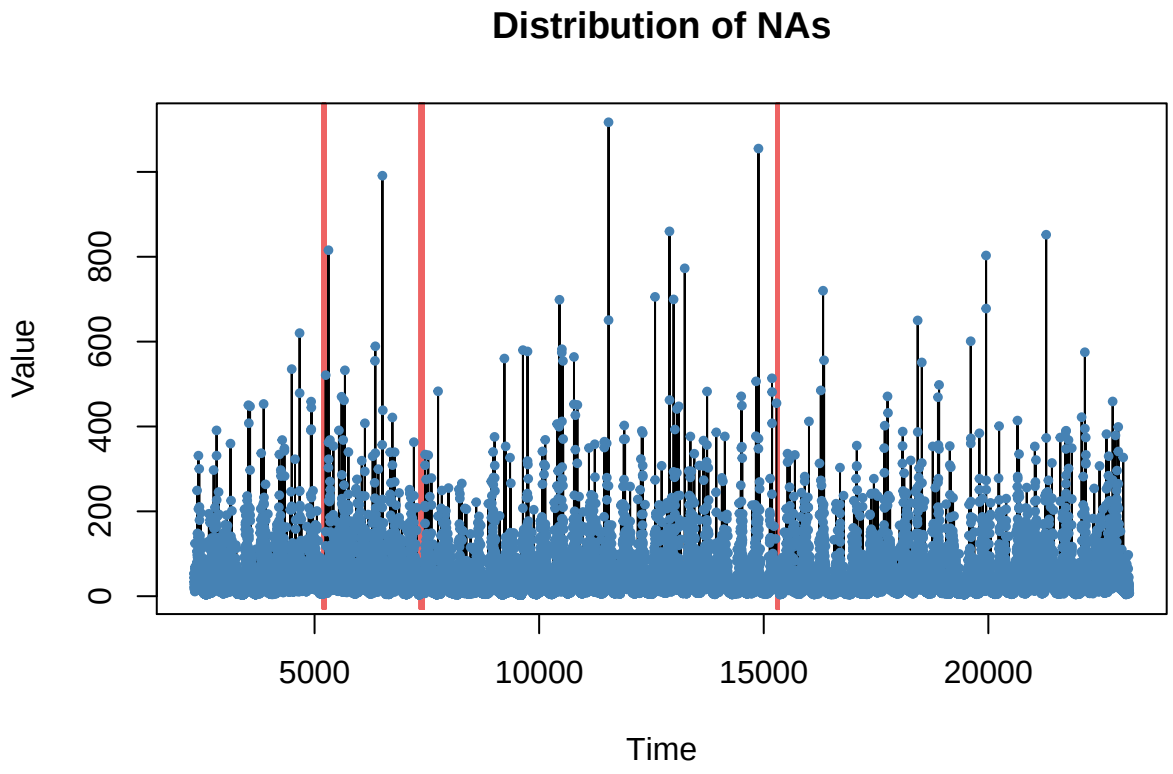
```
# 2 `heatwaves` as per the definition Unsurprisingly, they occur in June and
# August 2003
```

6.3.2 Exercise 2: Tyne river flow

```
tyne <- read.csv(file = "http://sma.epfl.ch/~lbelzile/math342/23001-Tyne_at_Bywell.csv",
    header = FALSE, sep = ",", skip = 16, col.names = c("time", "height", "flag"),
    as.is = TRUE, na.strings = "NA")[, 1:2]
library(imputeTS)
plot(which(is.na(tyne)), ylab = "Index of missing value within dataset", xlab = "Index within the missi
```




```
tyne_ts <- with(tyne, ts(data = height, start = c(as.numeric(substr(time[1],
    1, 4), lubridate::yday(time[1])), frequency = 365)))
plotNA.distribution(tyne_ts)
```




```
statsNA(tyne_ts)
```

```
[1] "Length of time series:"
[1] 20819
[1] "-------------------------"
[1] "Number of Missing Values:"
[1] 196
[1] "-------------------------"
[1] "Percentage of Missing Values:"
[1] "0.941%"
[1] "-------------------------"
[1] "Stats for Bins"
[1] "  Bin 1 (5205 values from 1 to 5205) :      154 NAs (2.96%)"
[1] "  Bin 2 (5205 values from 5206 to 10410) :      0 NAs (0%)"
[1] "  Bin 3 (5205 values from 10411 to 15615) :      42 NAs (0.807%)"
[1] "  Bin 4 (5204 values from 15616 to 20819) :      0 NAs (0%)"
[1] "-------------------------"
[1] "Longest NA gap (series of consecutive NAs)"
[1] "93 in a row"
[1] "-------------------------"
[1] "Most frequent gap size (series of consecutive NA series)"
[1] "93 NA in a row (occuring 1 times)"
[1] "-------------------------"
[1] "Gap size accounting for most NAs"
[1] "93 NA in a row (occuring 1 times, making up for overall 93 NAs)"
[1] "-------------------------"
[1] "Overview NA series"
[1] "  20 NA in a row: 1 times"
[1] "  22 NA in a row: 1 times"
[1] "  61 NA in a row: 1 times"
[1] "  93 NA in a row: 1 times"
```