timeseRies

Léo Belzile

version of 2018-09-28

Contents

Preliminary remarks

This is a web complement to MATH 342 (Time Series), a first regression course for EPFL mathematicians.

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 to download the program. The most popular graphical cross-platform front-end is RStudio Desktop.

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.

Chapter 1

Practical series 1

1.1 Exploratory Data Analysis

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 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. Many websites provide overview of exploratory data analysis (EDA).

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 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")`.

1.1.1 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.

```
# 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
- Mike West's datasets
- Ruey Tsay's datasets from the book *Analysis of Financial Time Series*
- Bo Li's paleoclimate datasets
- Kaggle datasets
- Don Percival's data page (navigate to Data tab)
- Carbon dioxide data
- Climate Research Unit
- NOAA

1.1.3 Time series objects and basic plots

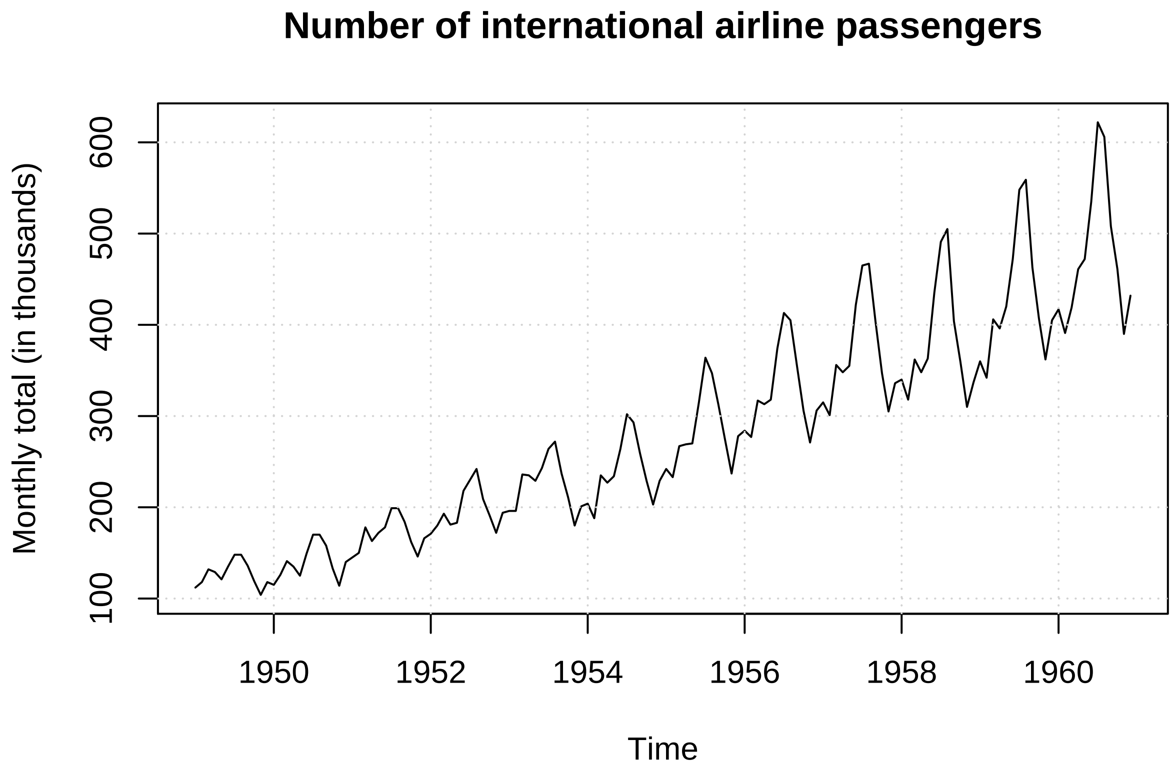
Objects in **R** are vectors by default, which have a type and attributes (vector is a type, `length` is an attribute of vectors). Some objects also inherit a class, such as `ts`. They inherit printing and plotting methods specific to the data class.

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.

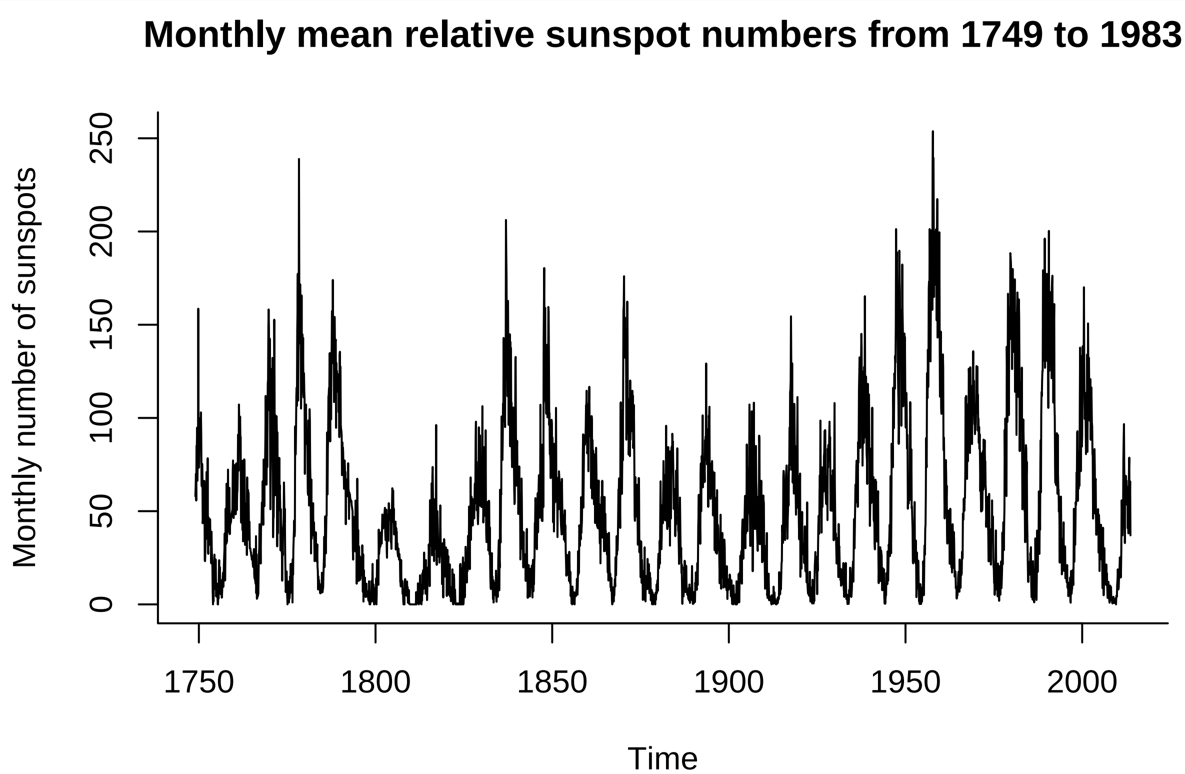
```
# AirPassenger dataset, lazy-loaded
class(AirPassengers)  #object of class 'ts'
```

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

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

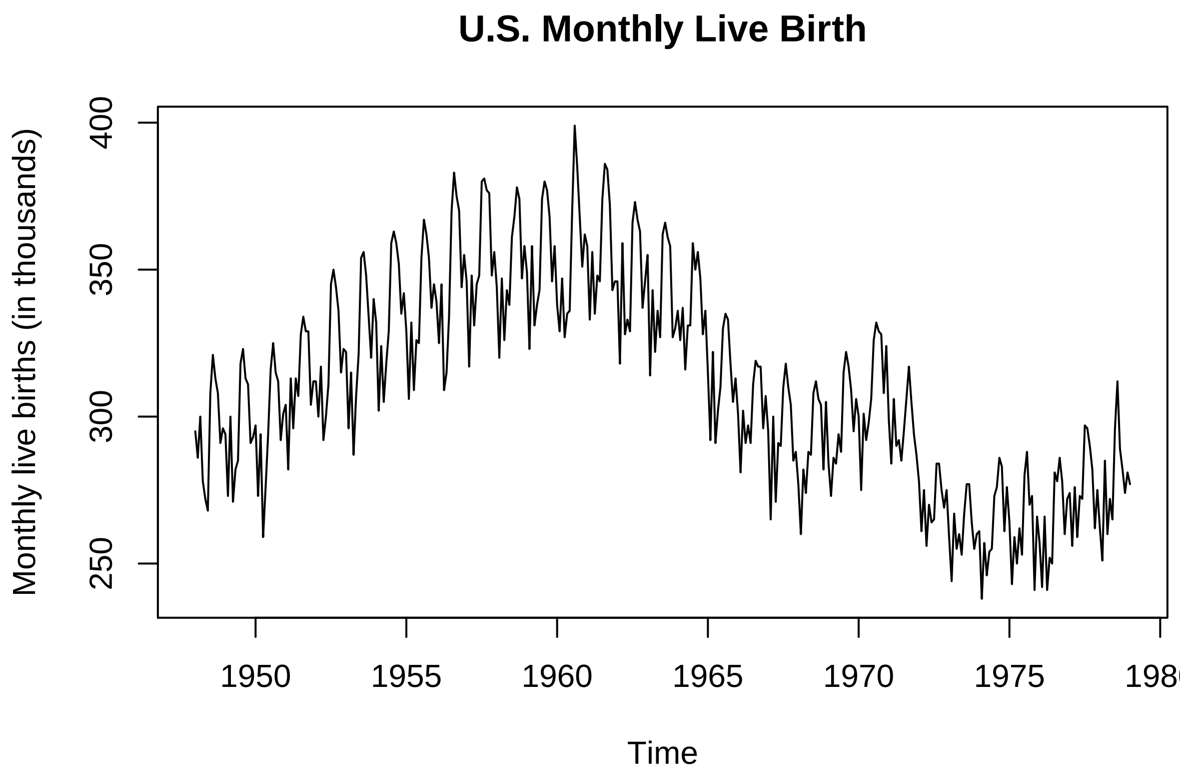



```r
`?`(sunspot.month)
plot(sunspot.month, ylab = "Monthly number of sunspots", main = "Monthly mean relative sunspot numbers
    bty = "l")
```




```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
```

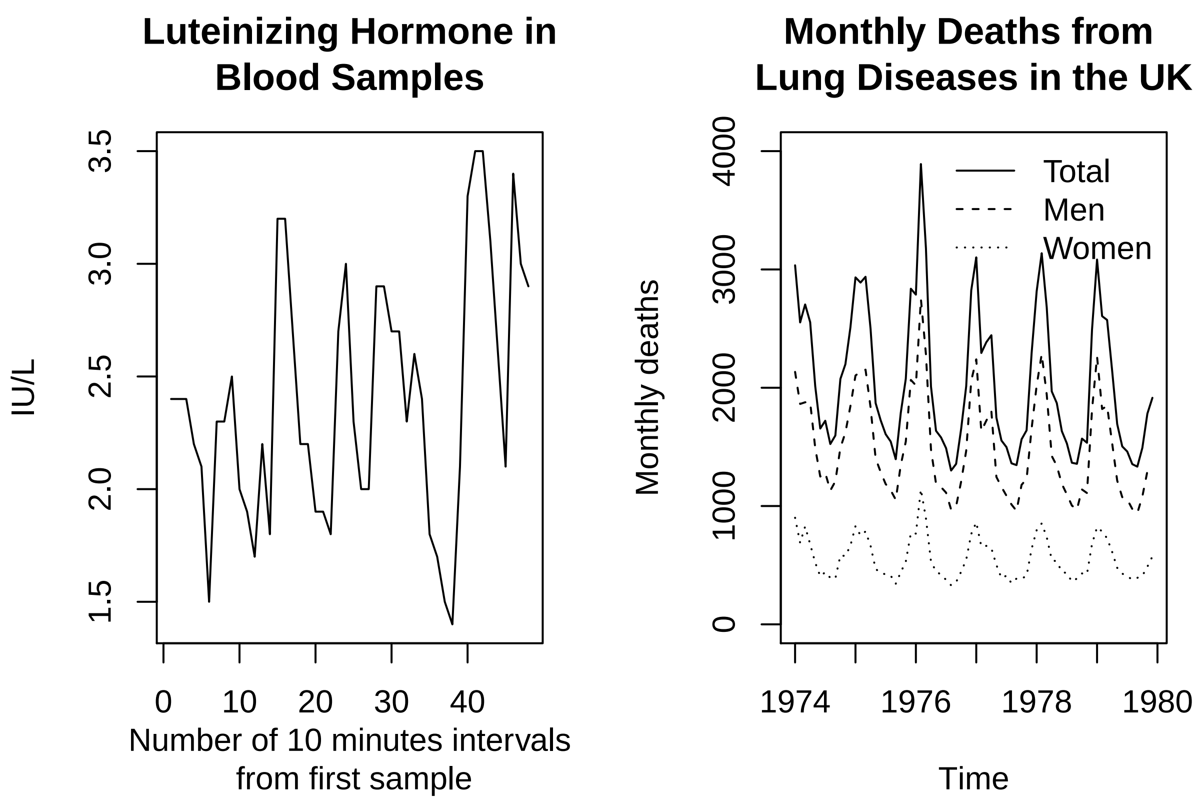
```
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1974 3035 2552 2704 2554 2014 1655 1721 1524 1596 2074 2199 2512
1975 2933 2889 2938 2497 1870 1726 1607 1545 1396 1787 2076 2837
1976 2787 3891 3179 2011 1636 1580 1489 1300 1356 1653 2013 2823
1977 3102 2294 2385 2444 1748 1554 1498 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
```

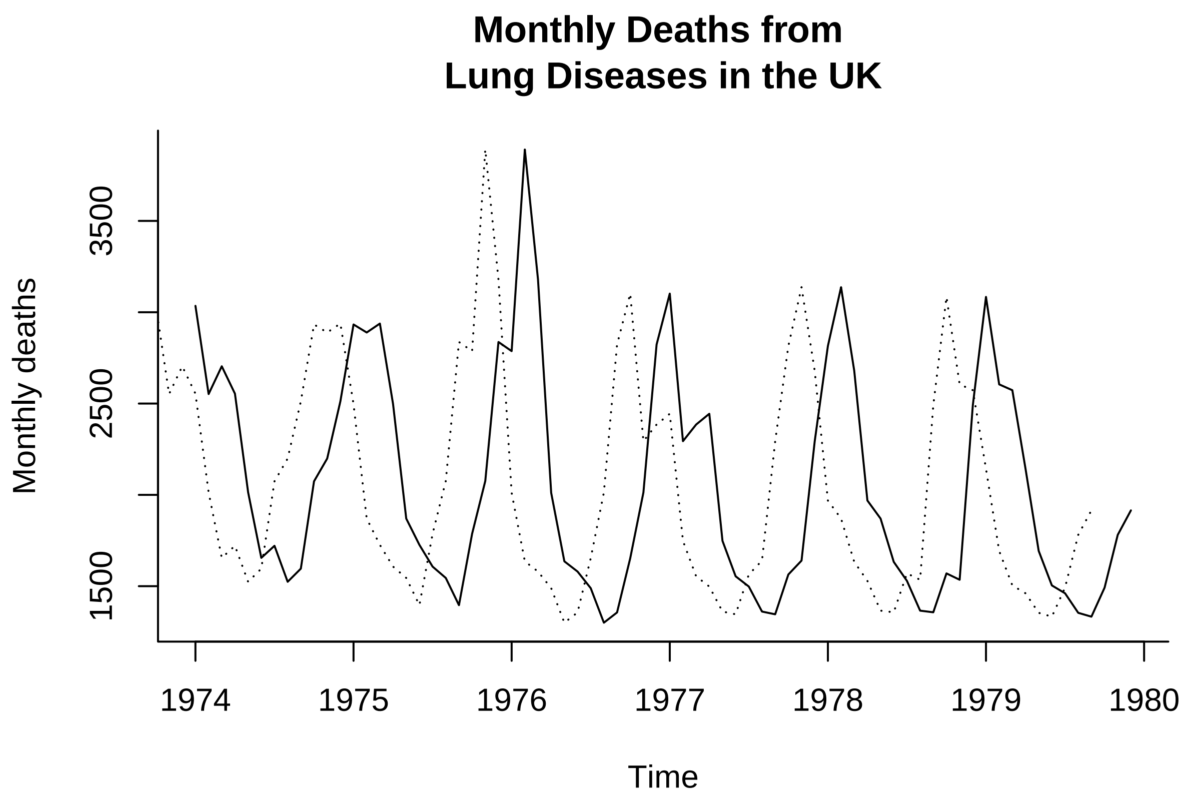
Above, you can see plots of `lh` and the three series on deaths. In the right-hand plot, the dashed series is for males, the dotted series for females and the solid line for the total.

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)
```

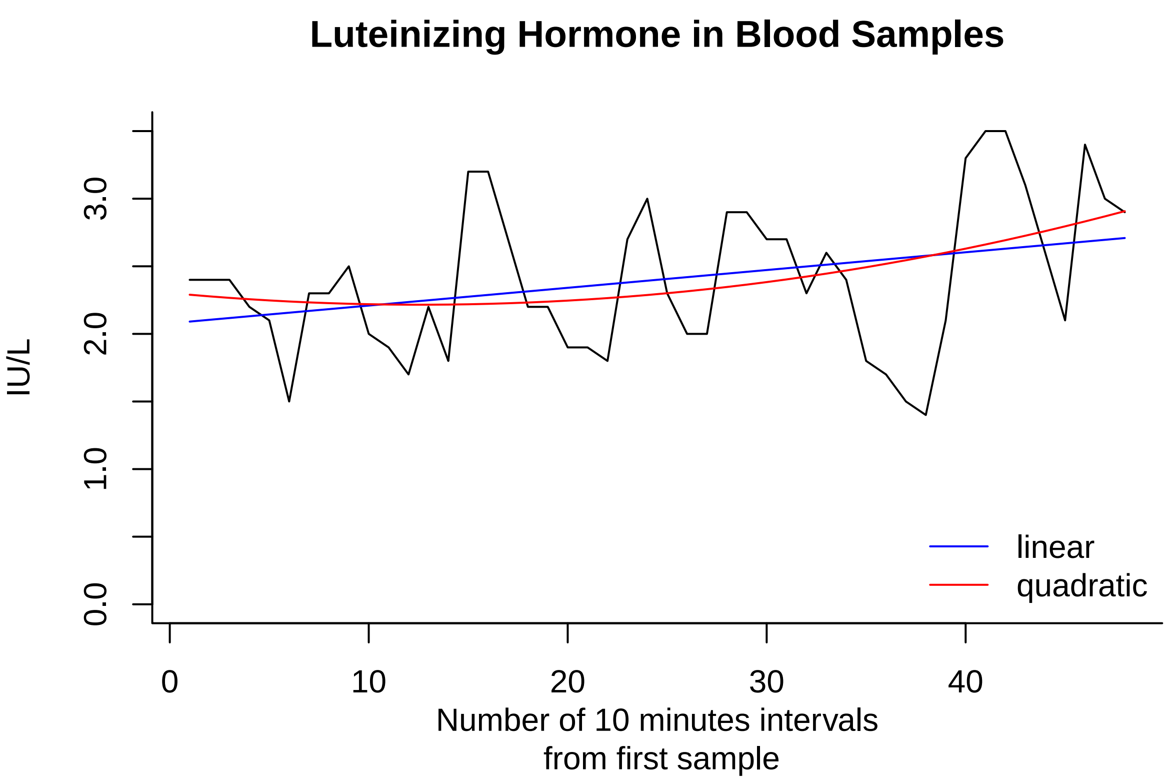
```
         Qtr1     Qtr2     Qtr3     Qtr4
1974 2763.667 2074.333 1613.667 2261.667
1975 2920.000 2031.000 1516.000 2233.333
1976 3285.667 1742.333 1381.667 2163.000
1977 2593.667 1915.333 1401.667 1832.333
1978 2877.000 1824.000 1417.333 1865.333
1979 2754.000 1780.000 1382.667 1729.333
```

```
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 inte
    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")
```



YOUR TURN

1.2.0.1 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?

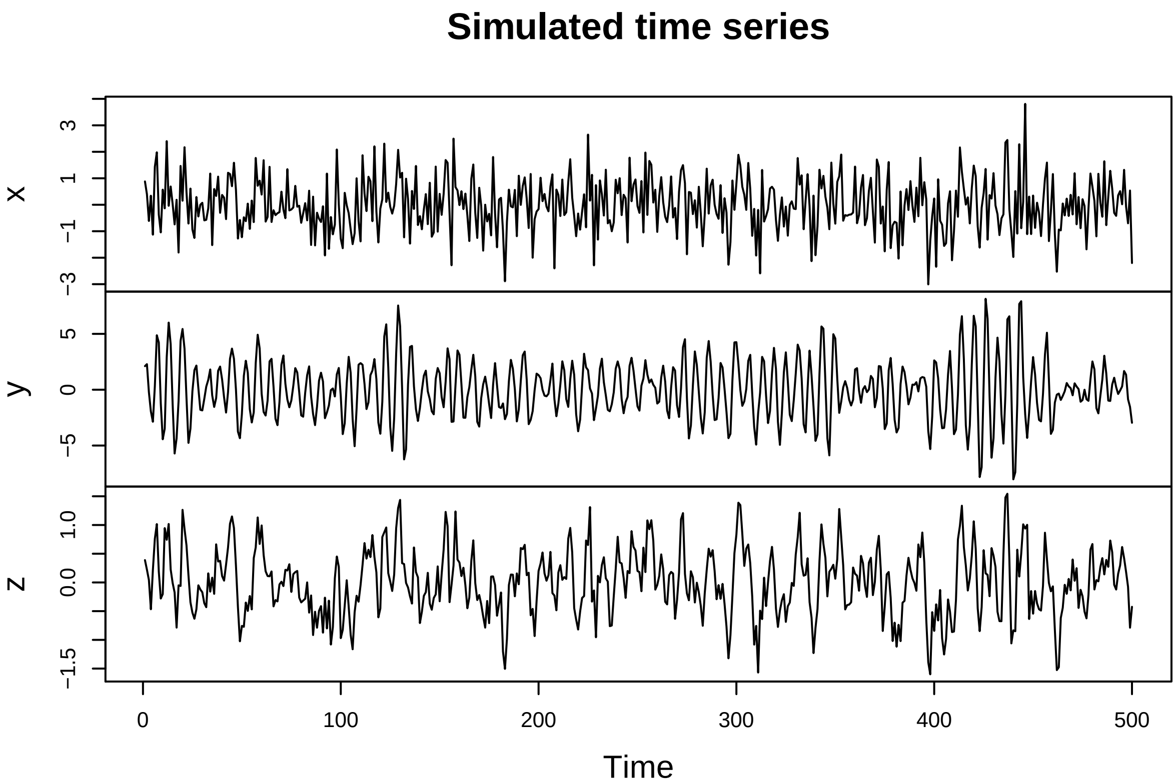
1.3 Second order stationarity

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

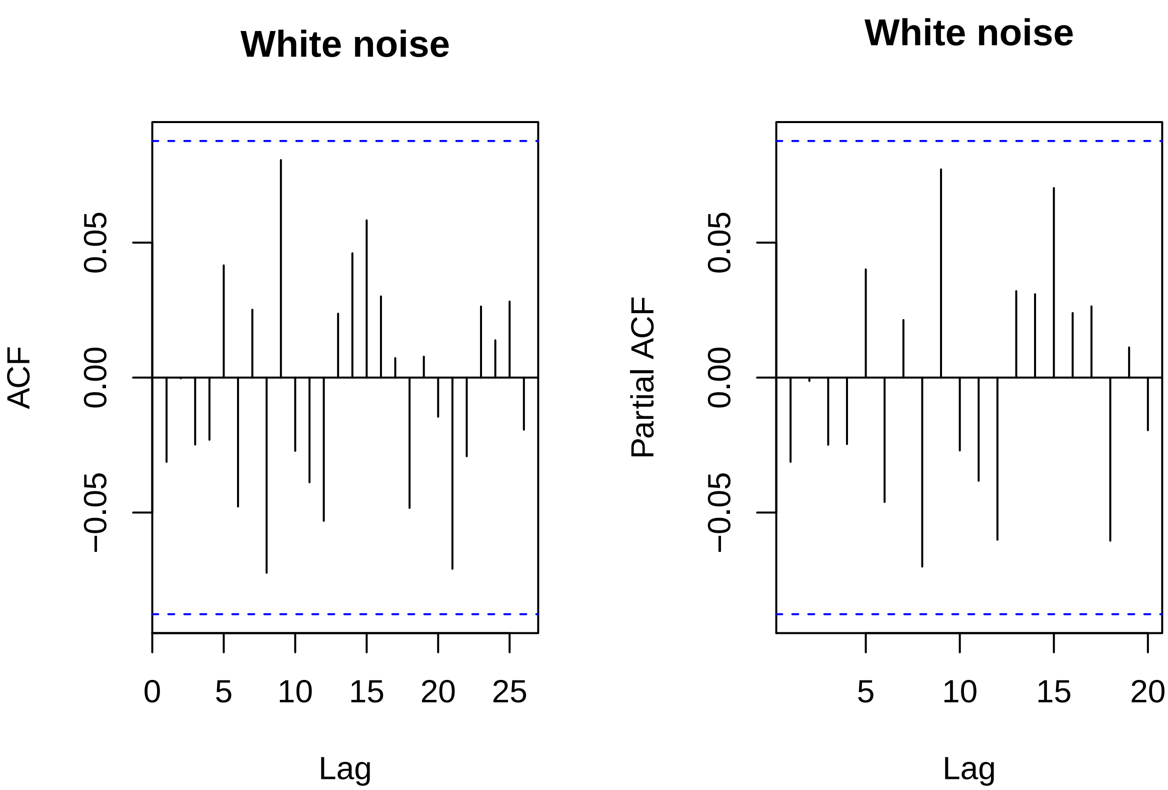
$$c_t = n^{-1} \sum_{s=\max(1,-t)}^{\min(n-t,n)} [X_{s+t} - \bar{X}][X_s - \bar{X}], \quad r_t = c_t/c_0.$$

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

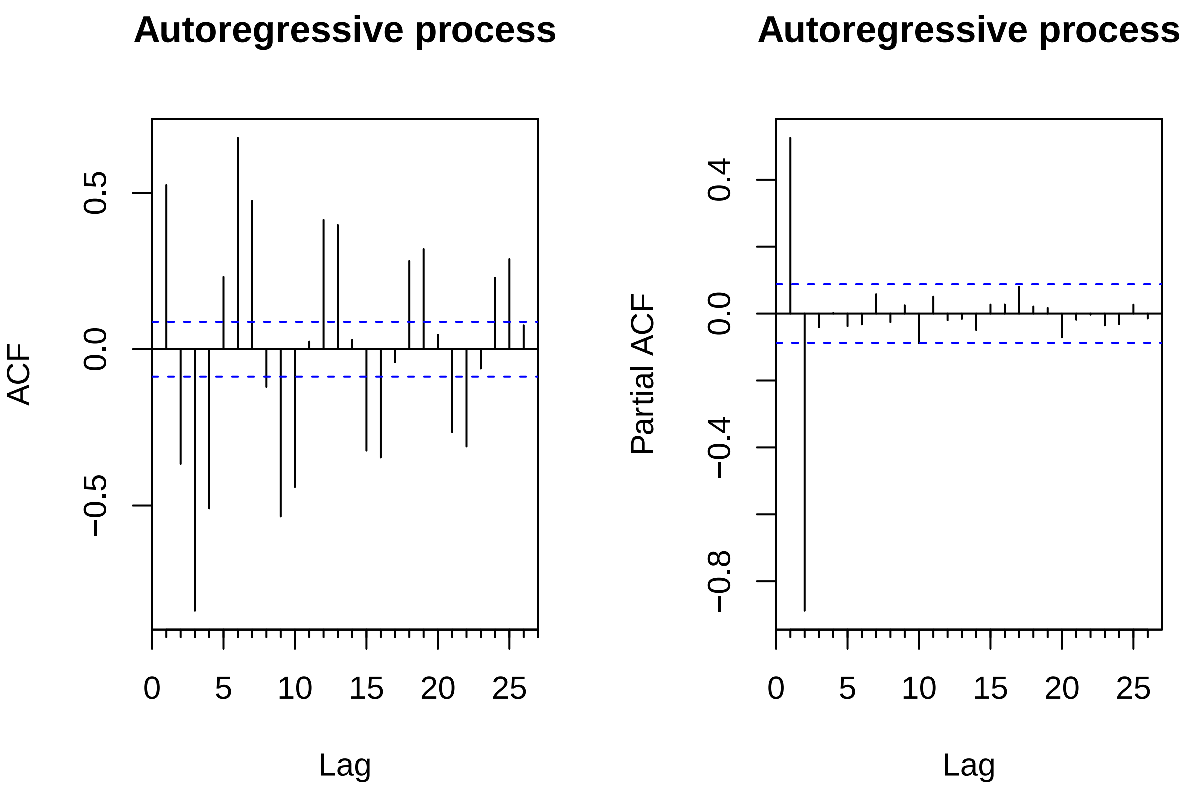
$$c_{ij}(t) = n^{-1} \sum_{s=\max(1,-t)}^{\min(n-t,n)} [X_i(s+t) - \bar{X}_i][X_j(s) - \bar{X}_j].$$

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)
```

```r
    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)
```

YOUR TURN

1.3.1 Exercise 2. SP500 daily returns

1. Download the dataset using the following command

```r
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.

```r
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.

```r
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)
```

YOUR TURN

1.4.1 Exercise 3. Simulated data

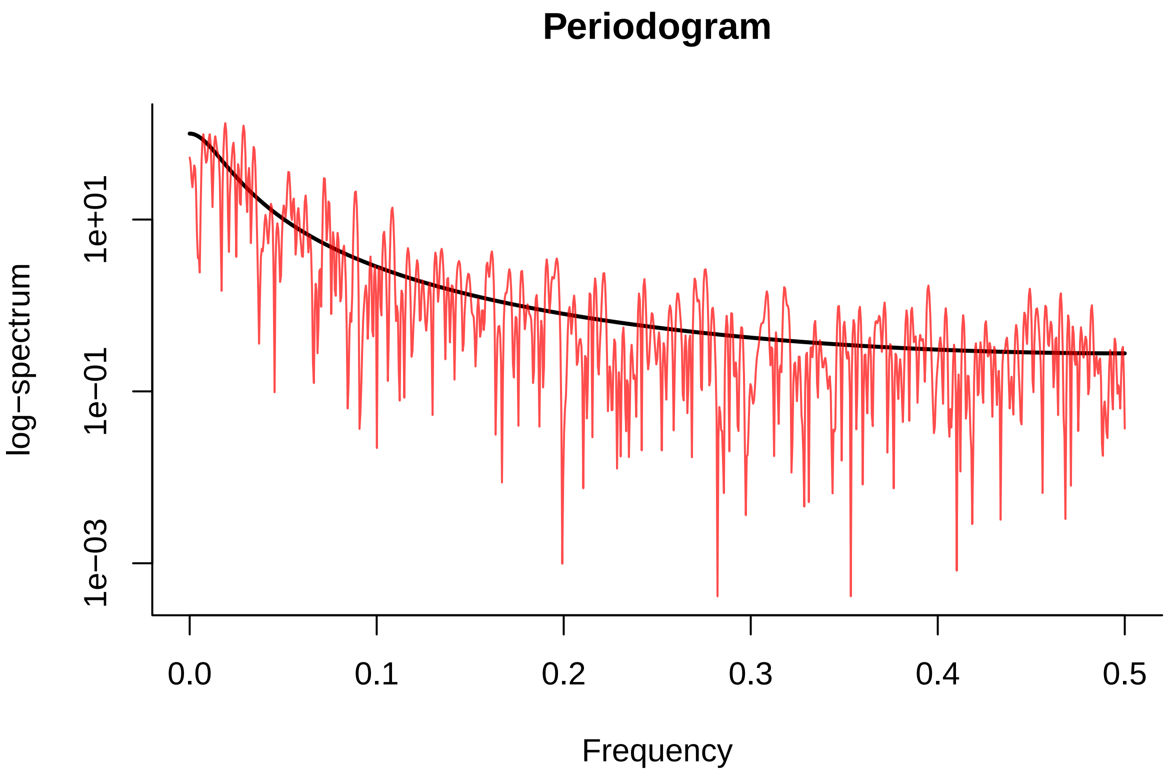
1. Simulate 500 observations from an AR(1) process with parameter values $\alpha \in \{0.1, 0.5, 0.9, 0.99\}$.
2. Repeat for MA processes of different orders. There is no restriction on the coefficients of the latter for stationarity, unlike the AR process.
3. Sample from an ARCH(1) process with Gaussian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

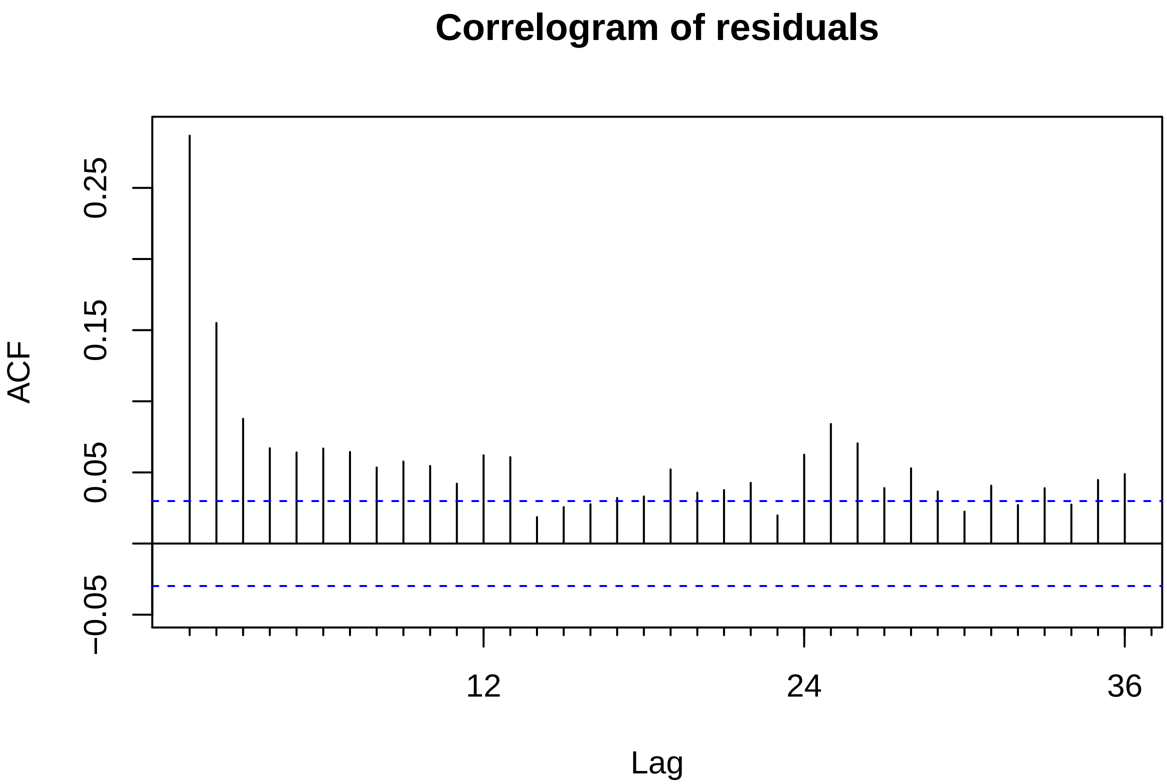
Now let us investigate the dataset in a regression context. Since it is an irregular time series, we use `zoo` rather than `ts`.

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

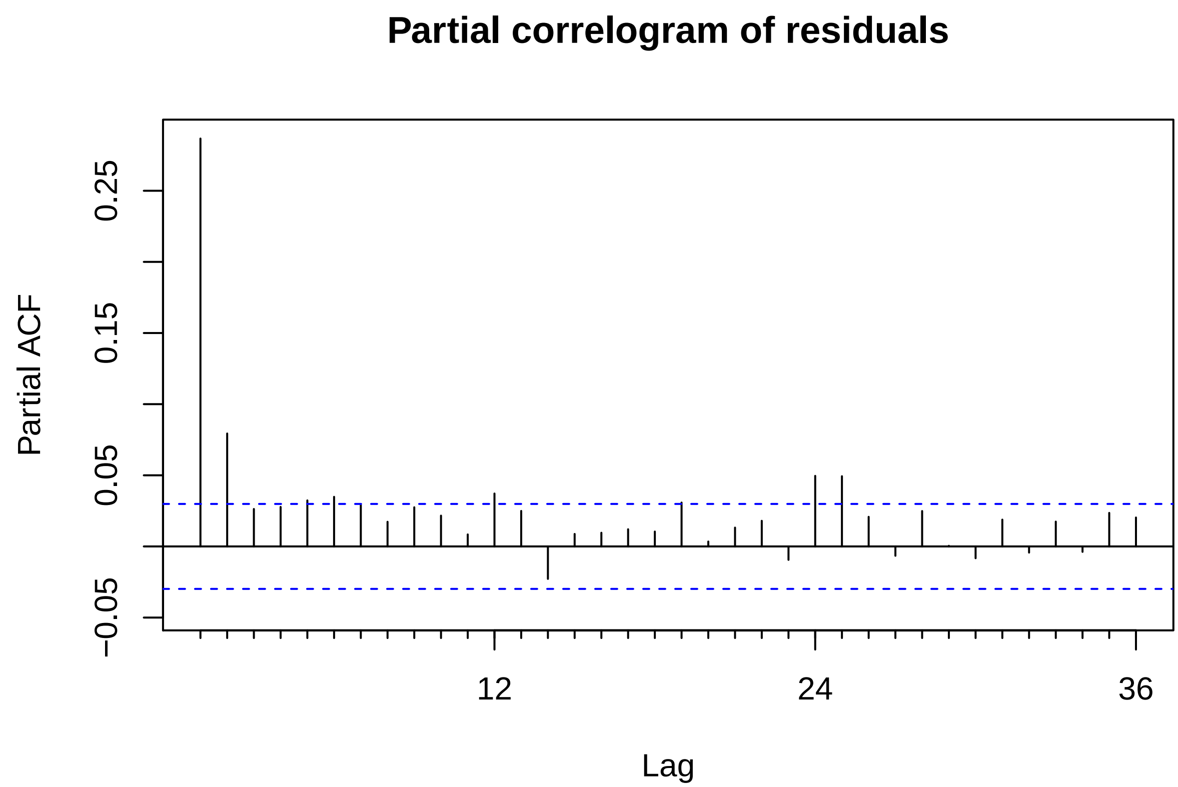
```
# Convert to time series object Create a time object using `seq`, then make
# into `yearmon` The latter has the same internal representation as `ts`
# with frequency
time <- zoo::as.yearmon(seq.Date(from = as.Date("1659/01/01"), length.out = prod(dim(cet)),
    by = "month"))
CET_ts <- zoo::zoo(c(t(cet)), 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")
```



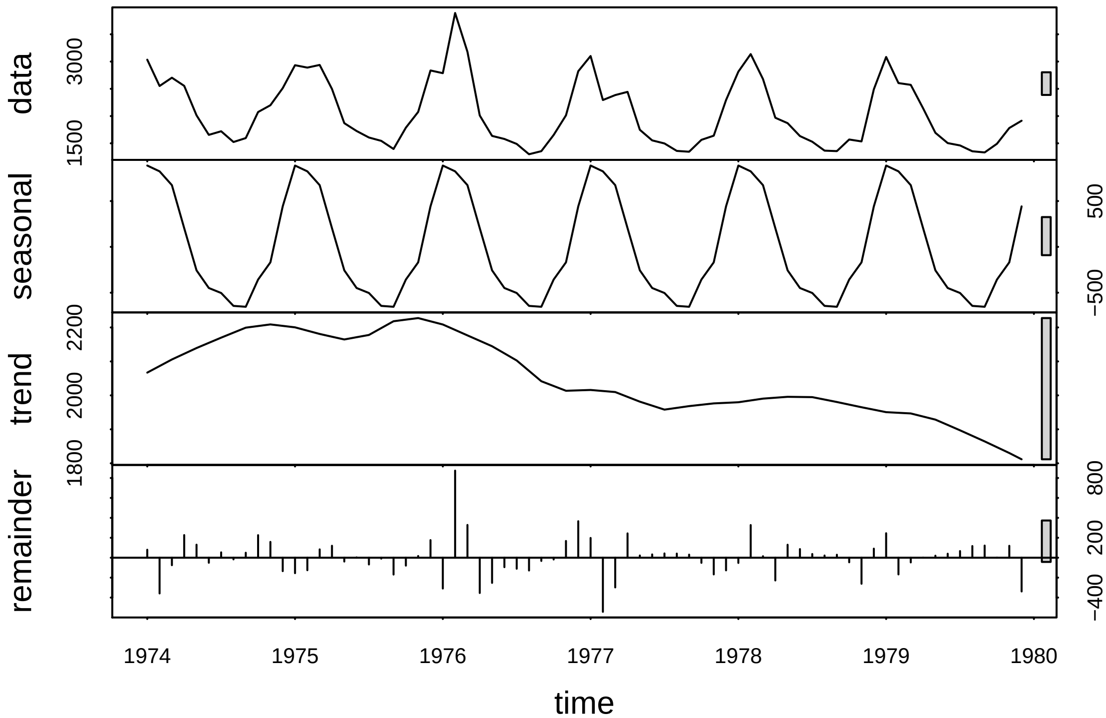

```r
forecast::Pacf(resid(ts1_b), main = "Partial correlogram of residuals")
```



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)
```



YOUR TURN

1.6.0.1 Exercise 4. Mauna Loa Atmospheric CO_2 Concentration

1. Load and plot the CO_2 dataset from NOAA. 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.

Chapter 2

Practical series 1 (solutions)

2.0.1 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?

```
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)
```

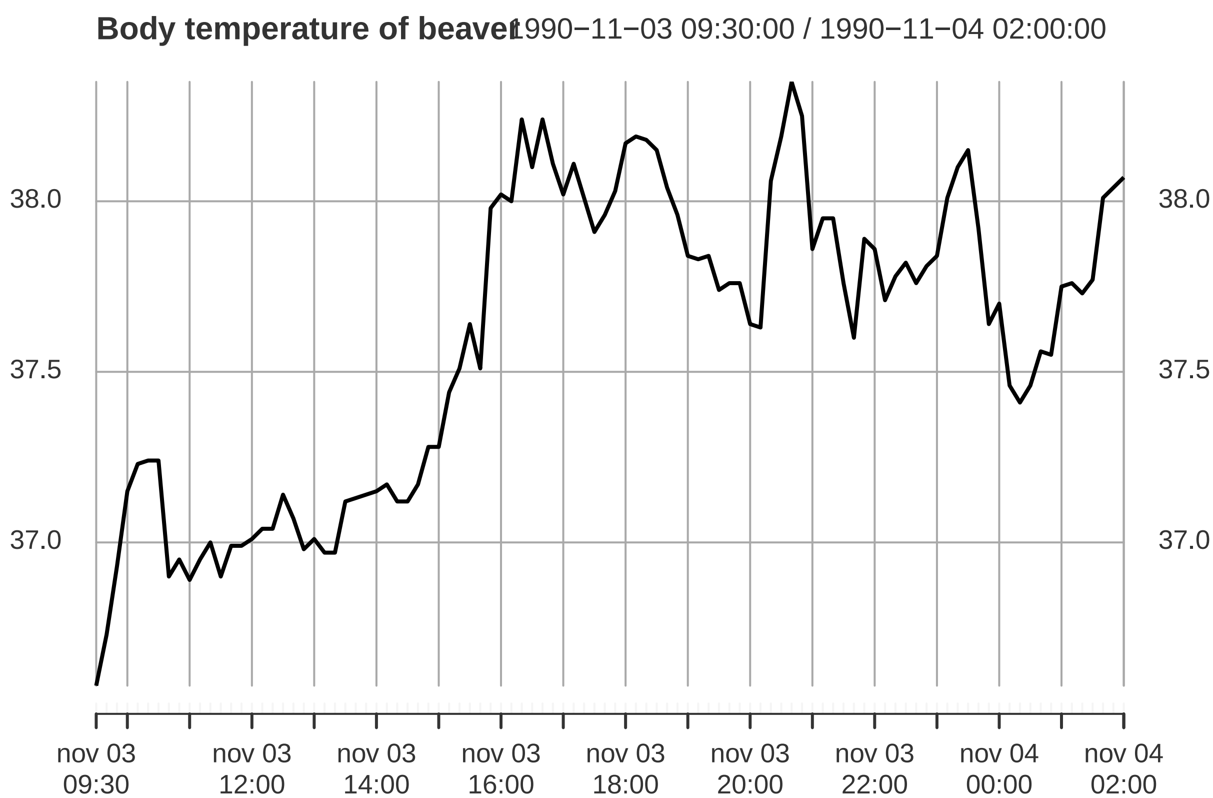
```
  day time  temp activ     hours
1 307  930 36.58     0  9.500000
```

```
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
```

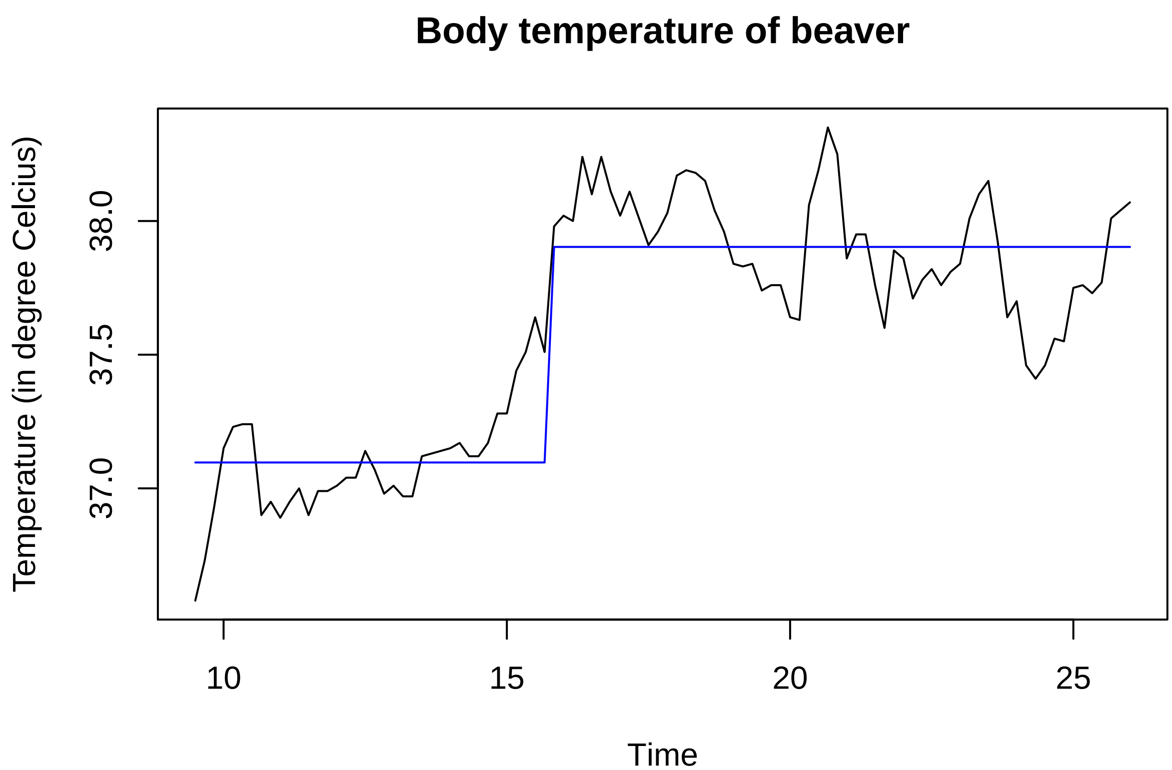
```
tail(beav2)
```

```
    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)")
```

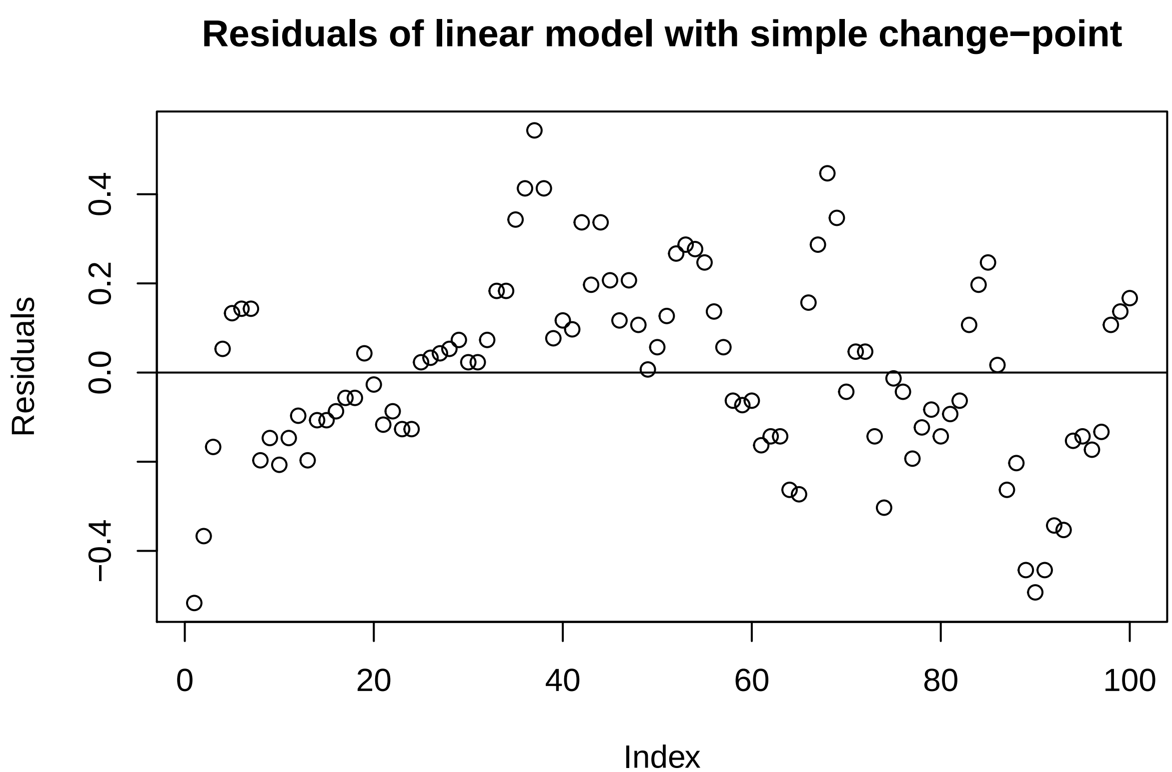



```
# Vanilla ts - works ok for regular time series
temp <- ts(beav2[, "temp"], start = 9.5, frequency = 6)
plot(temp, main = "Body temperature of beaver", ylab = "Temperature (in degree Celcius)")
lin_mod <- lm(temp ~ as.factor(activ), data = beav2)
lines(beav2[, "hours"], fitted(lin_mod), col = "blue")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

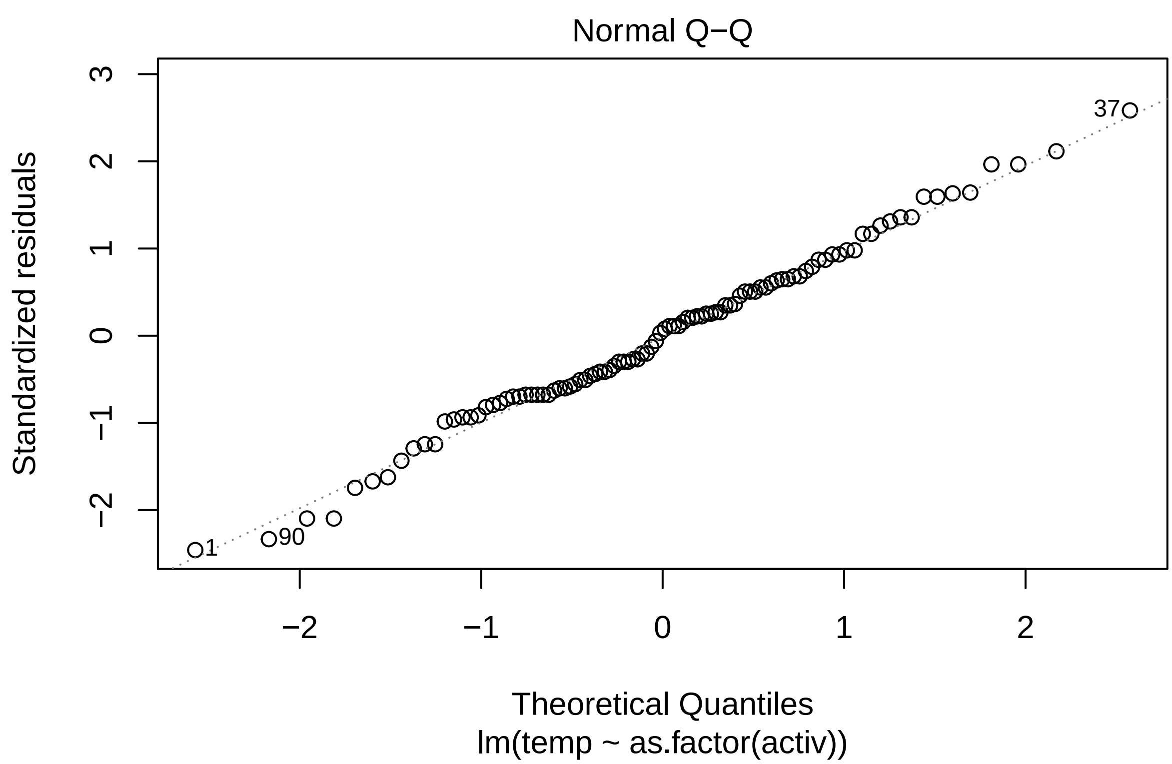



```r
# Some trend remaining in the first part, before time
```

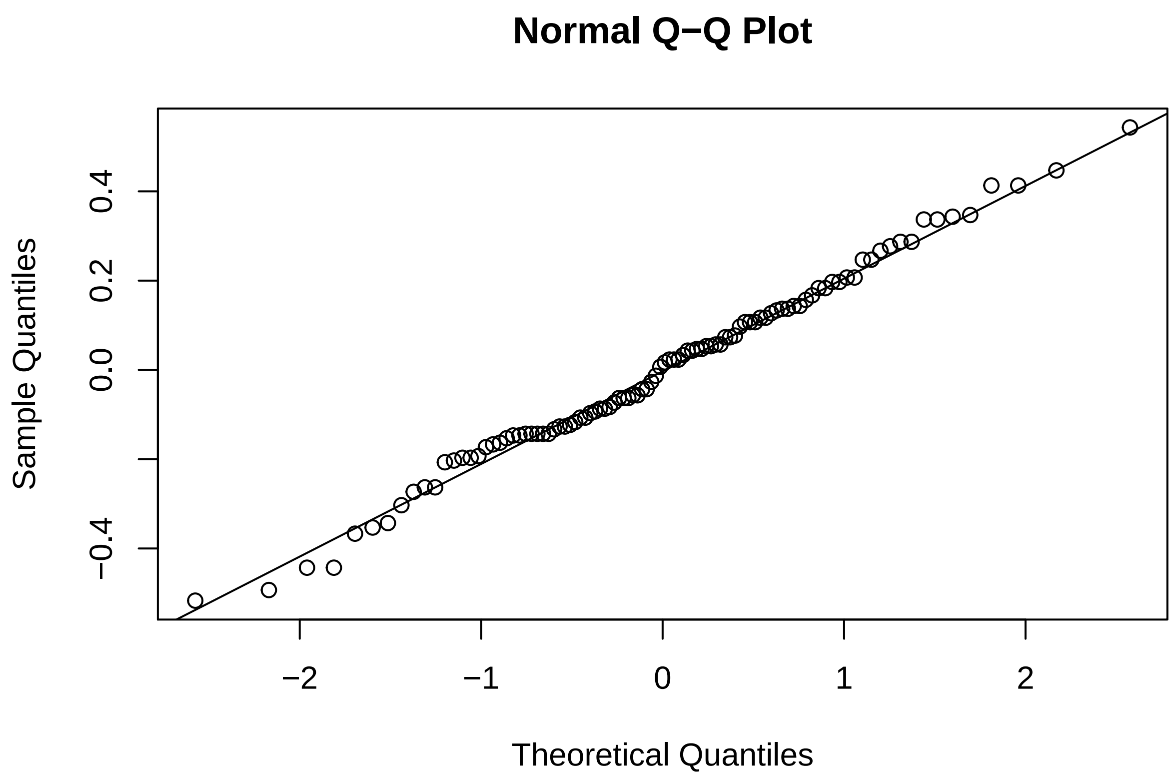
```r
plot(residuals(lin_mod), ylab = "Residuals", main = "Residuals of linear model with simple change-point
abline(h = 0)
```



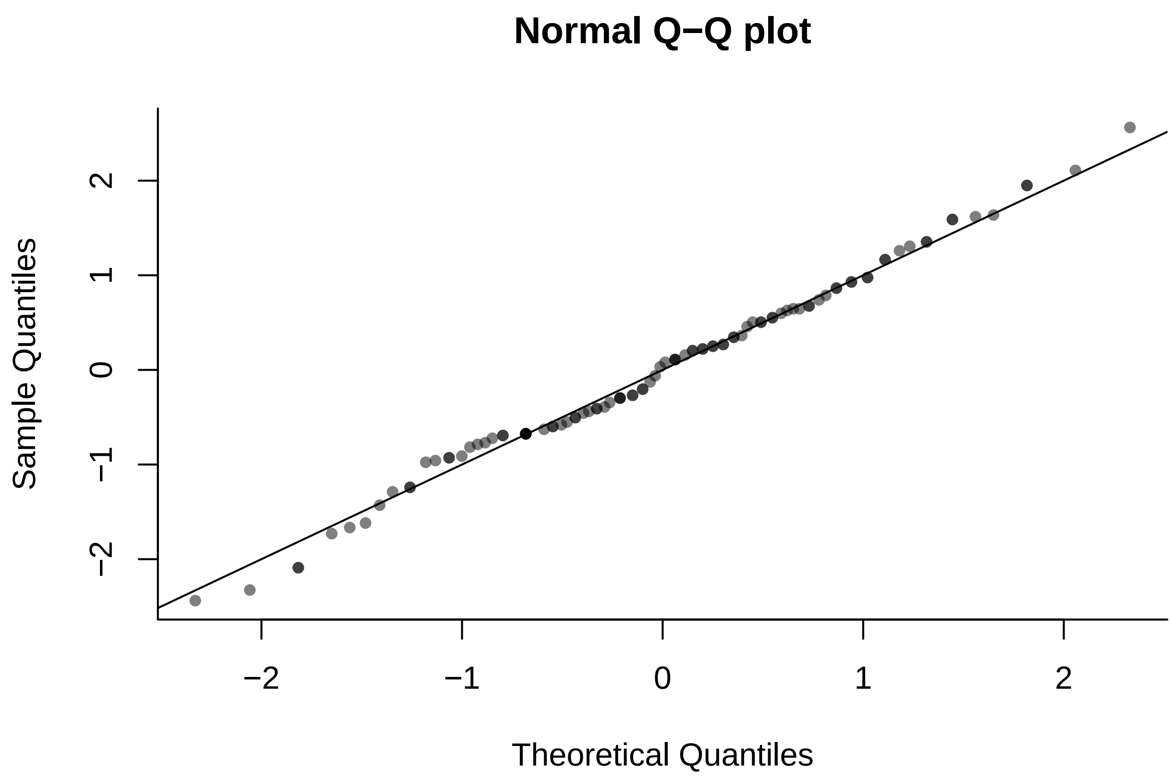
```
# Q-Q plot (1) with output from lm
plot(lin_mod, which = 2)
```



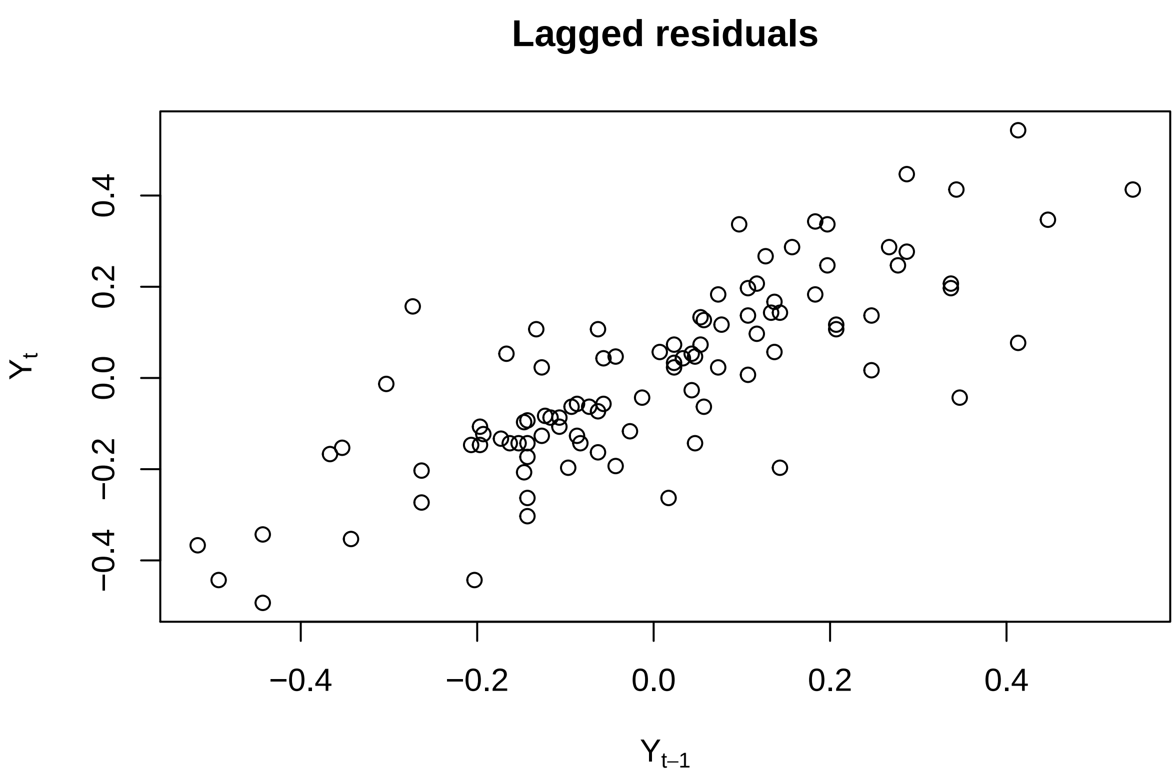

```
# (2) with qqnorm and tentative line with quartiles
qqnorm(residuals(lin_mod))
qqline(residuals(lin_mod))
```




```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")
```



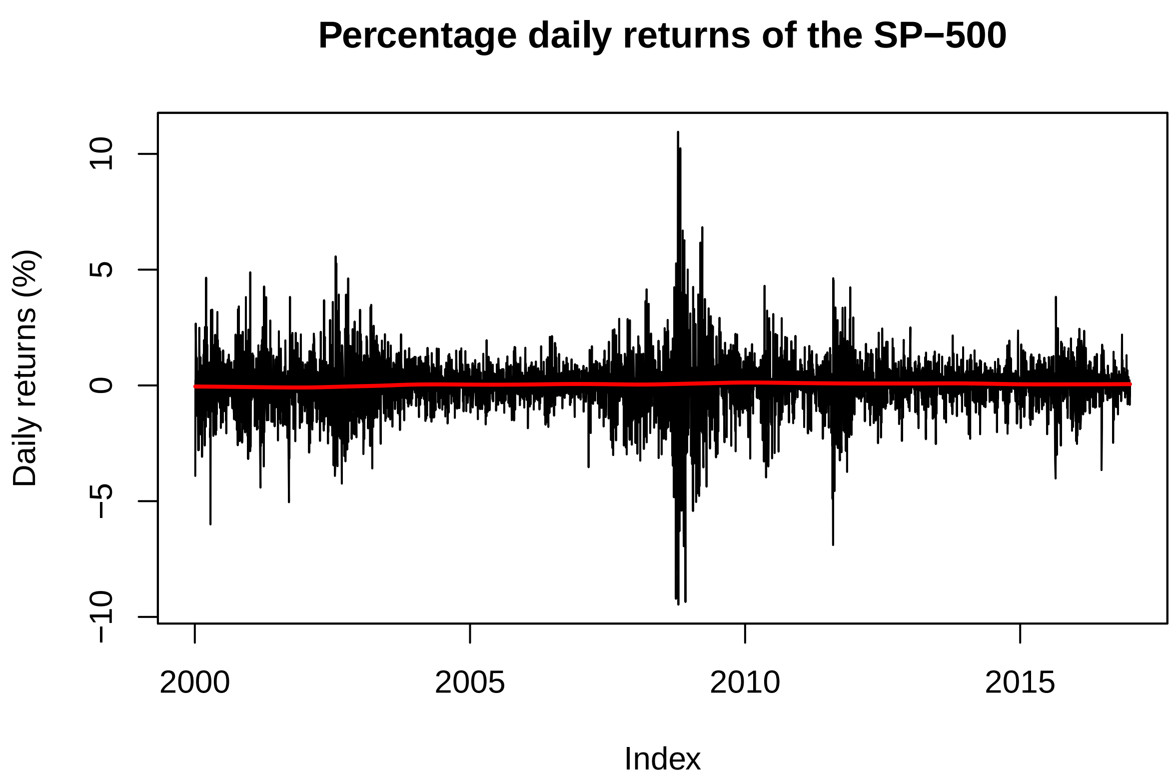
2.0.2 Exercise 2. SP500 daily returns

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?

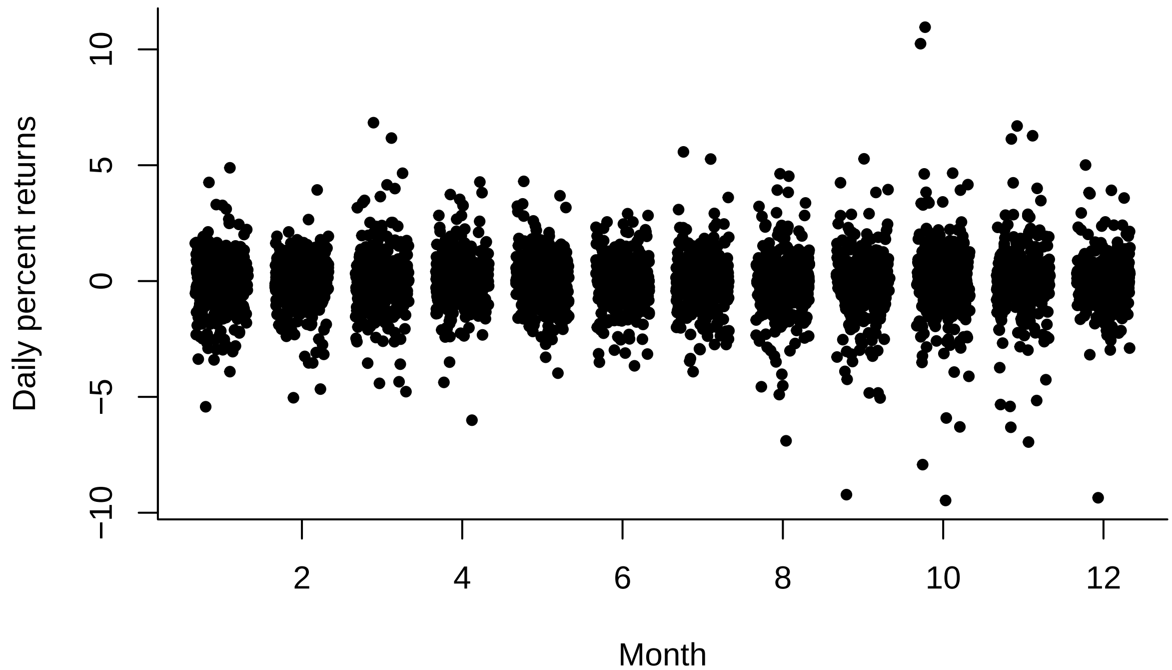
```
sp500 <- tseries::get.hist.quote(instrument = "^GSPC", start = "2000-01-01",
    end = "2016-12-31", quote = "AdjClose", provider = "yahoo", origin = "1970-01-01",
    compression = "d", retclass = "zoo")
```

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

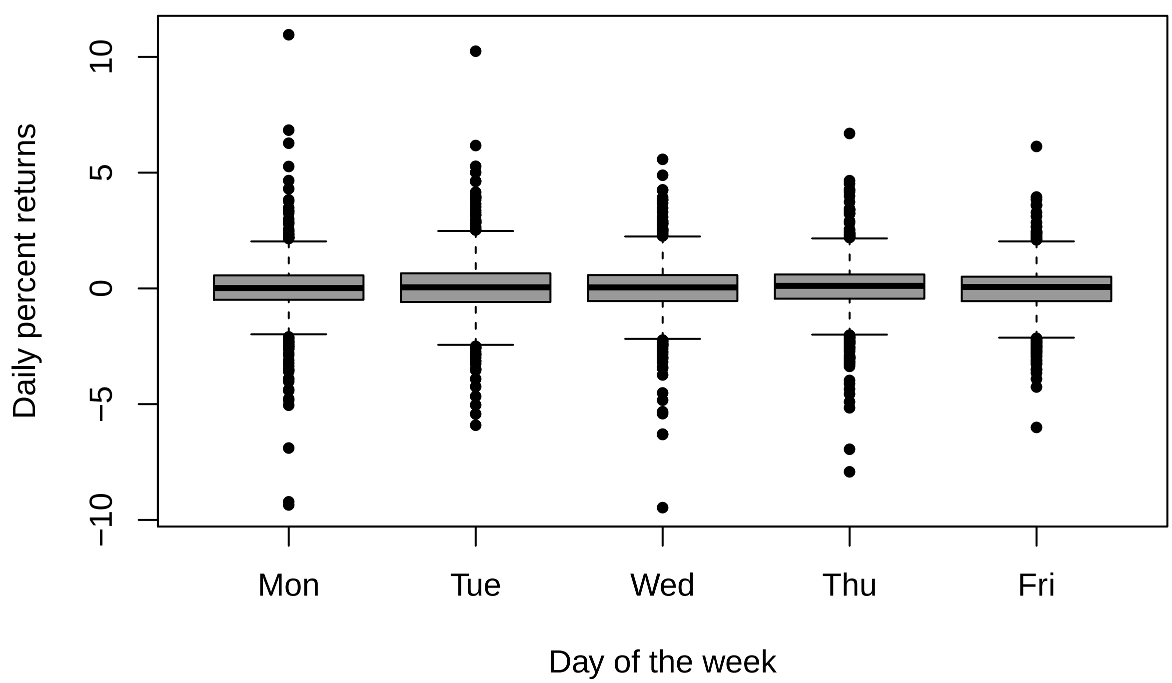
```
library(xts)
library(lubridate)
# Daily return in percentage
spret <- 100 * diff(log(sp500))
plot(spret, ylab = "Daily returns (%)", main = "Percentage daily returns of the SP-500")
# local trend to see if there is any evidence of non-zero trend
lines(index(spret), lowess(spret, f = 1/5)$y, col =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



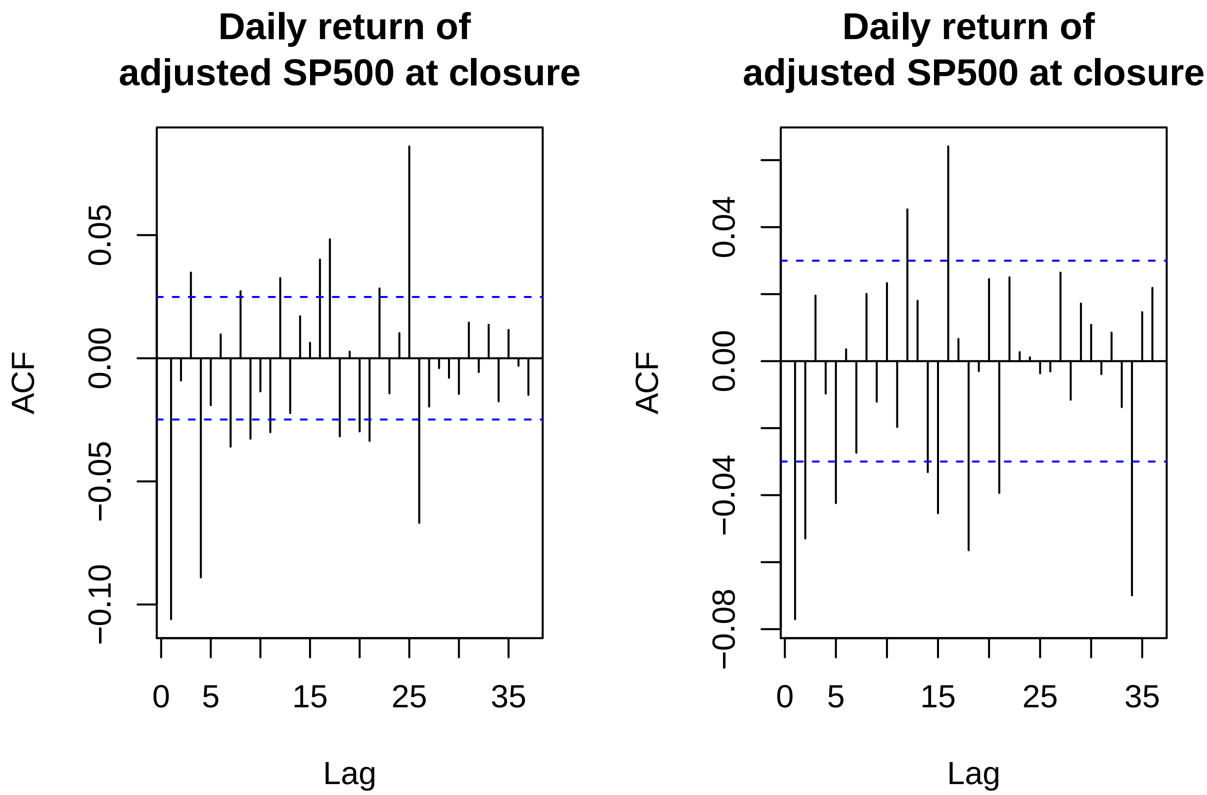

```
# Volatility as function of month
plot(jitter(month(spret), amount = 1/3), spret, pch = 20, ylab = "Daily percent returns",
    xlab = "Month", bty = "l")
```




```
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)
```




```r
dev.off()
```

```
null device
          1
```

```r
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)
```

2.0.3 Exercise 3. Simulated data

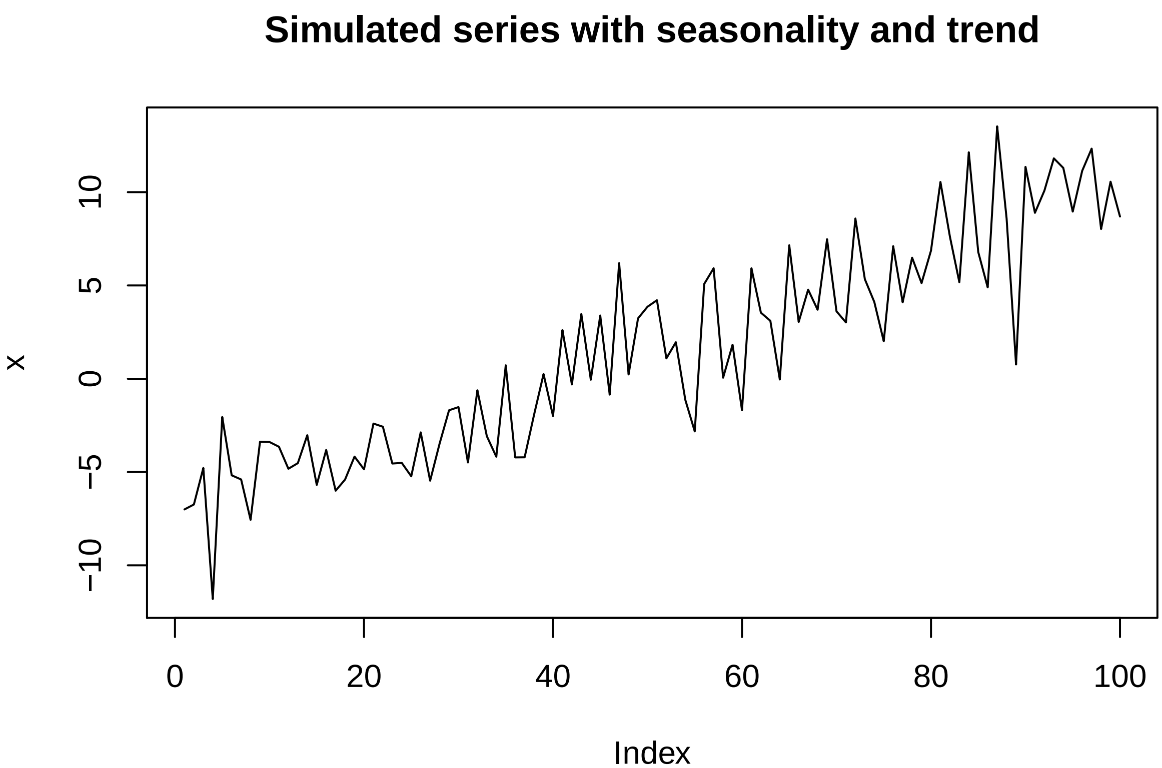
The first 5 parts of the question are straightforward and left to the reader.

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.

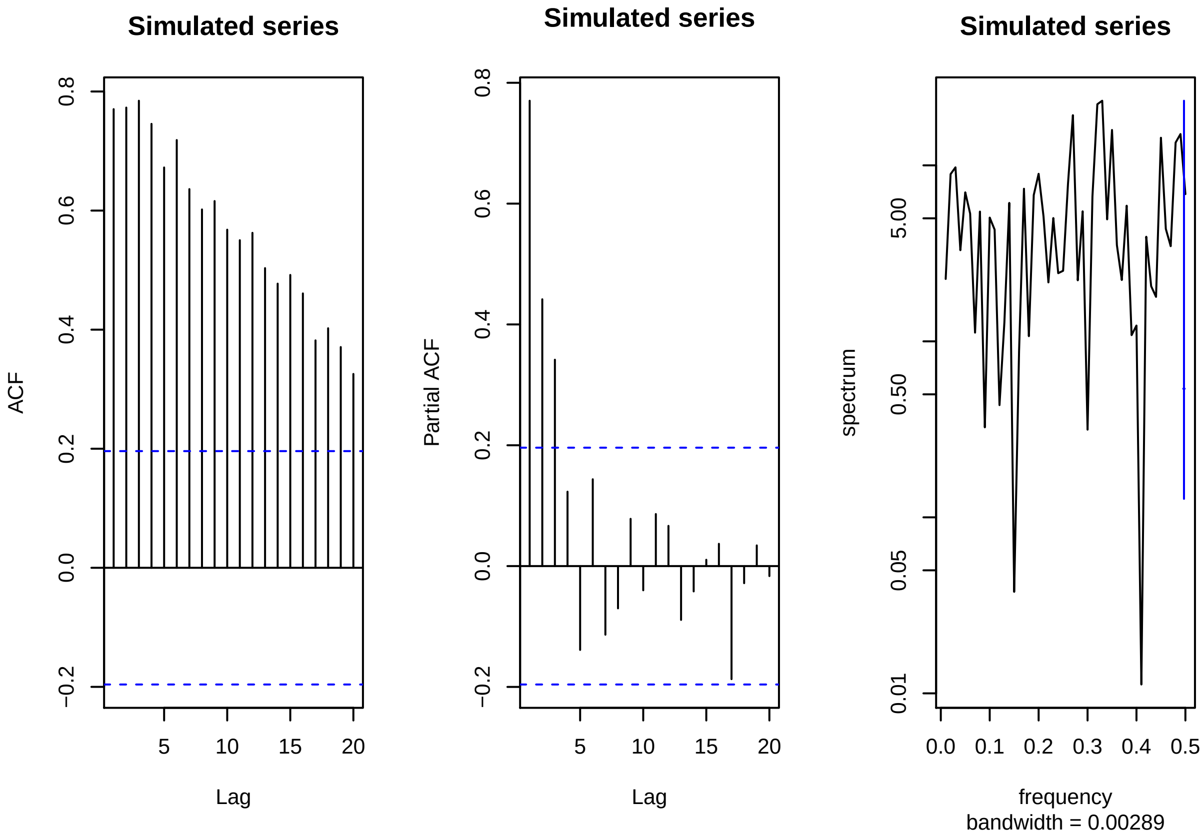
```
n <- 100
tim <- scale(1:n)
x <- 5 * tim + cos(2 * pi * tim/n) + rnorm(n, sd = 3)
plot(x, type = "l", main = "Simulated series with seasonality and trend")
```



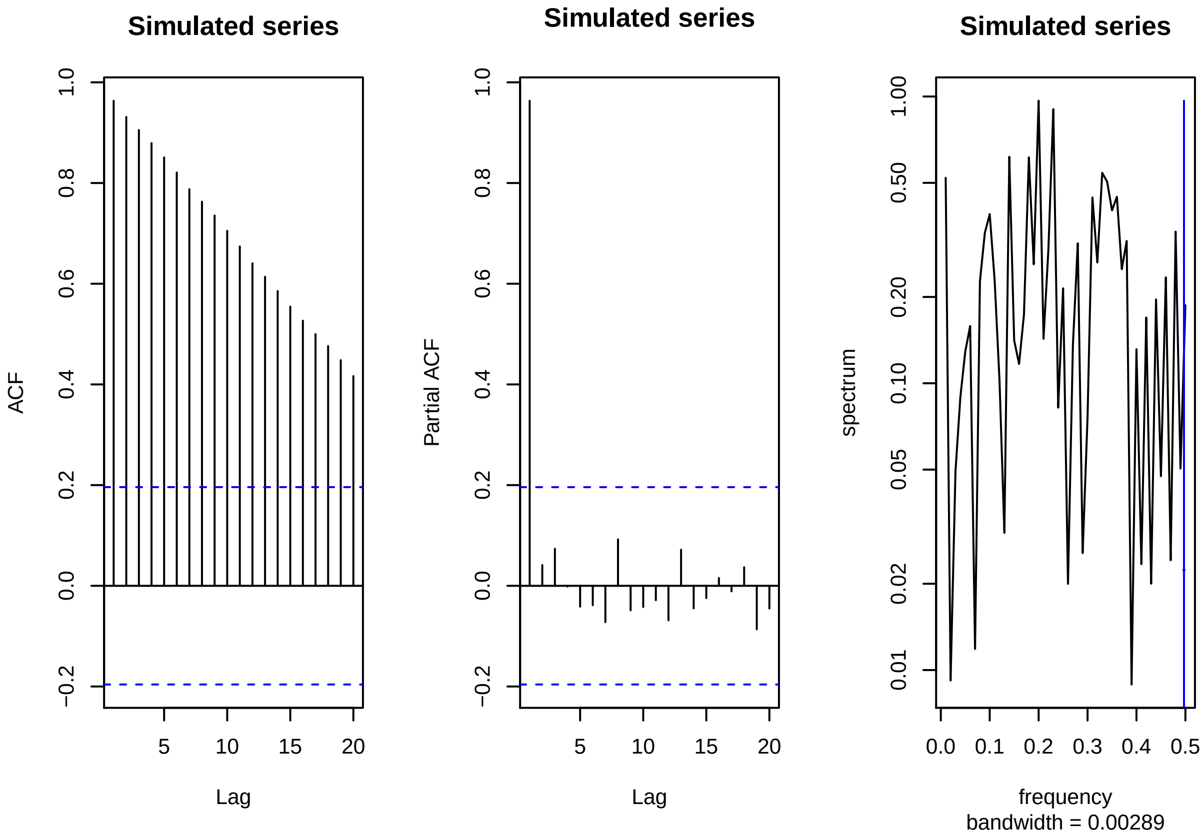

```
Box.test(x, type = "L")
```

```

    Box-Ljung test

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

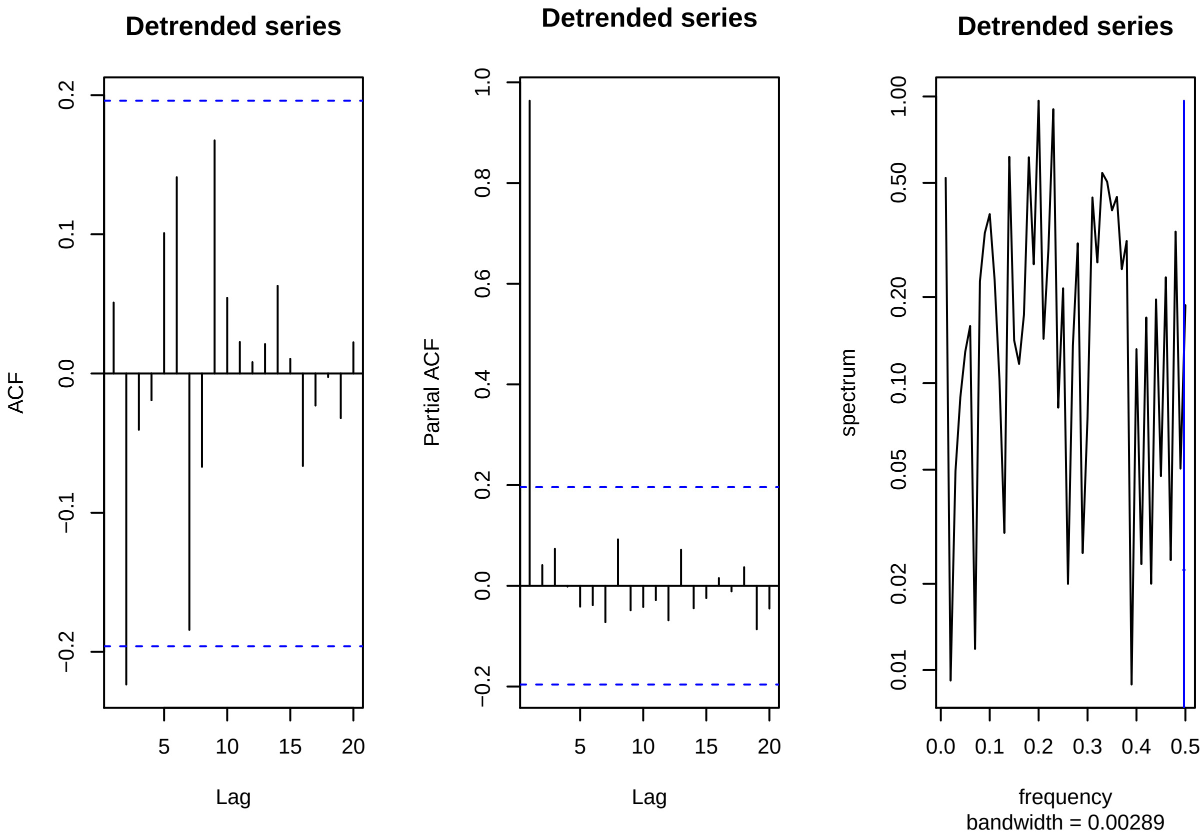
```
par(mfrow = c(1, 3))  # plots side by side
TSA::acf(x, main = "Simulated series")
pacf(x, main = "Simulated series")
spectrum(x, main = "Simulated series")
```



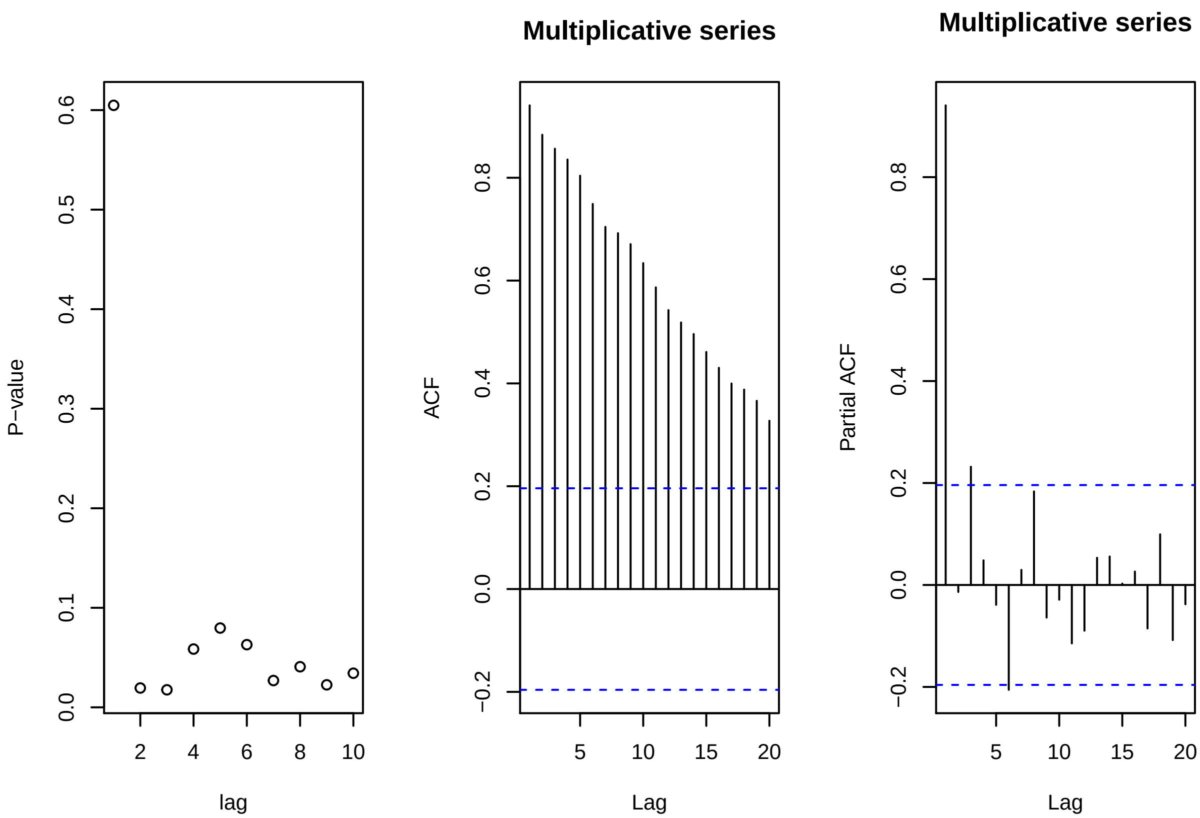

```r
# Nothing in spectrum, persistence in the (p)acf
x <- 5 * tim + cos(2 * pi * tim/n) + rnorm(n, sd = 0.5)
TSA::acf(x, main = "Simulated series")
pacf(x, main = "Simulated series")
spectrum(x, main = "Simulated series")
```




```
# 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
```

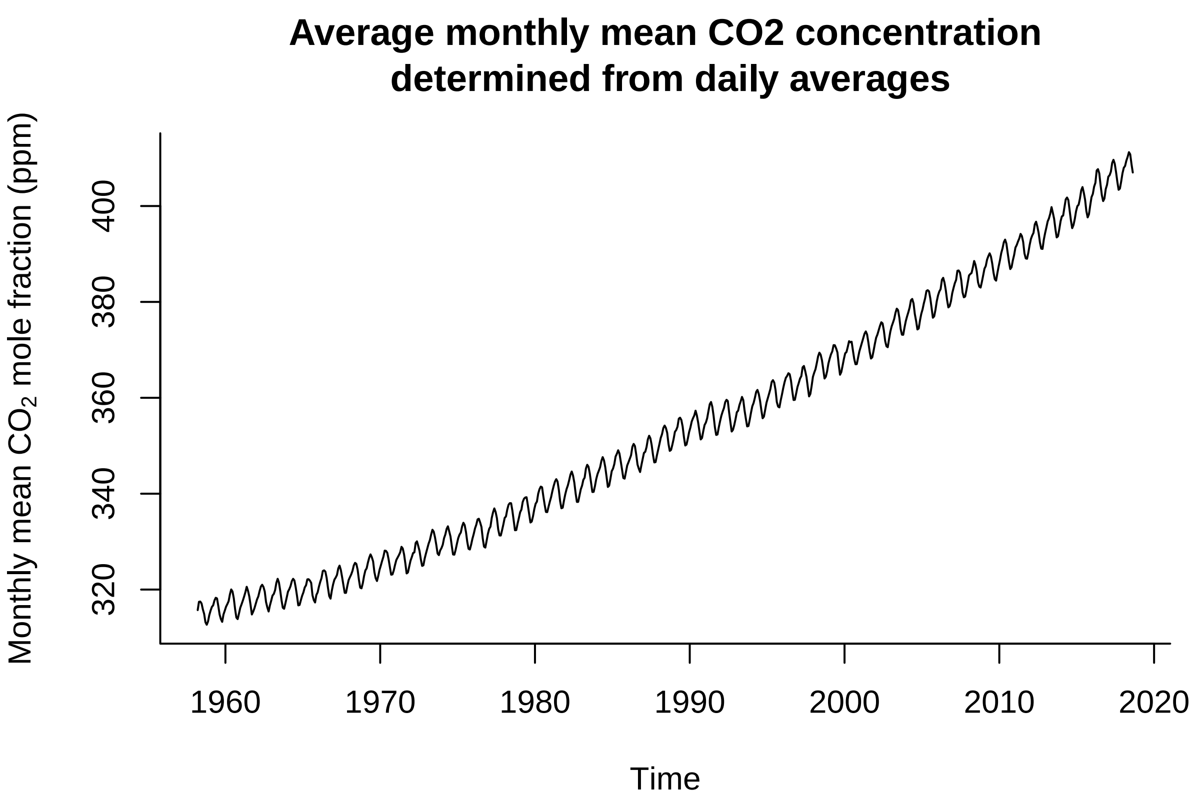
```
graphics.off()
# Now large impact on spectrum, and nonlinear features!
plot(mult, main = "Multiplicative series with lognormal margins", ylab = "",
    xlab = "Time")
# Note that decompose has an option type='multiplicative' for seasonal
# components
```

2.0.4 Exercise 4: Mauna Loa Atmospheric CO_2 Concentration

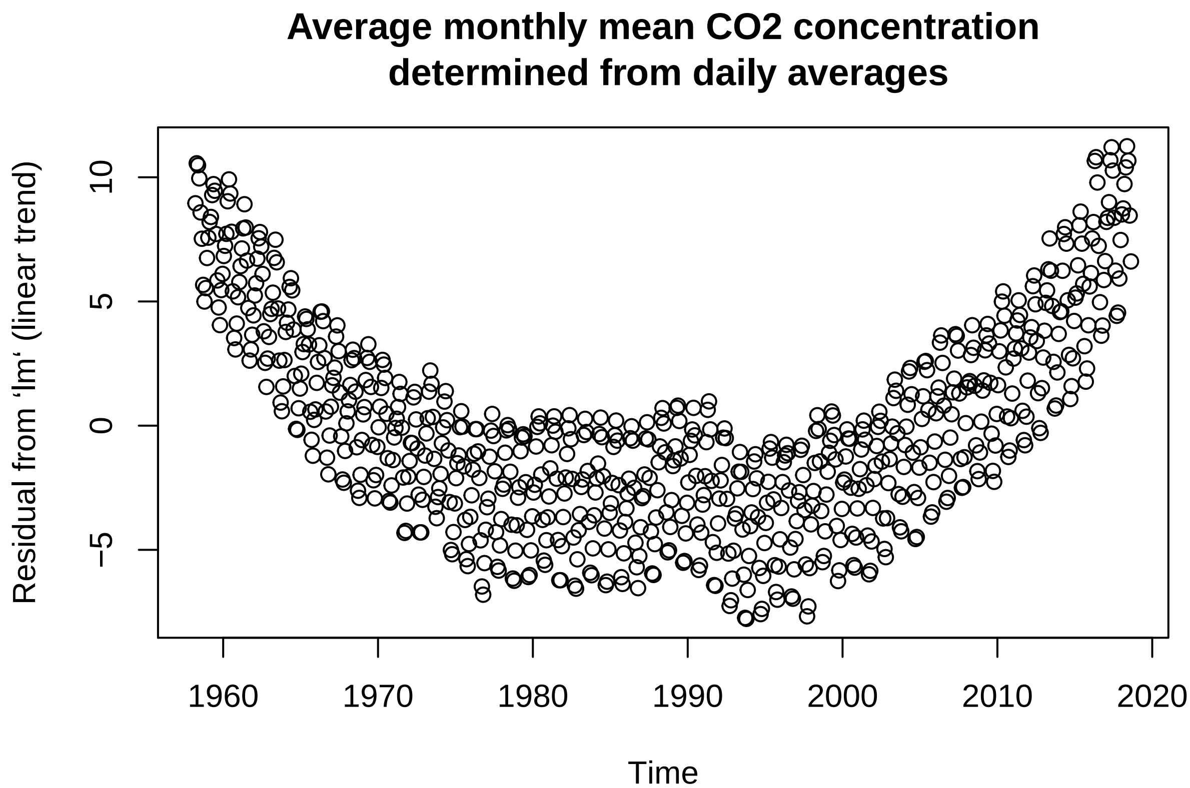
1. Load and plot the CO_2 dataset from NOAA. Pay special attention to the format, missing values, the handling of string and the description. Use `?read.table` for help, and look carefully at arguments `file`, `sep`, `na.strings`, `skip` and `stringsAsFactors`. From now on, we will work with the complete 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.

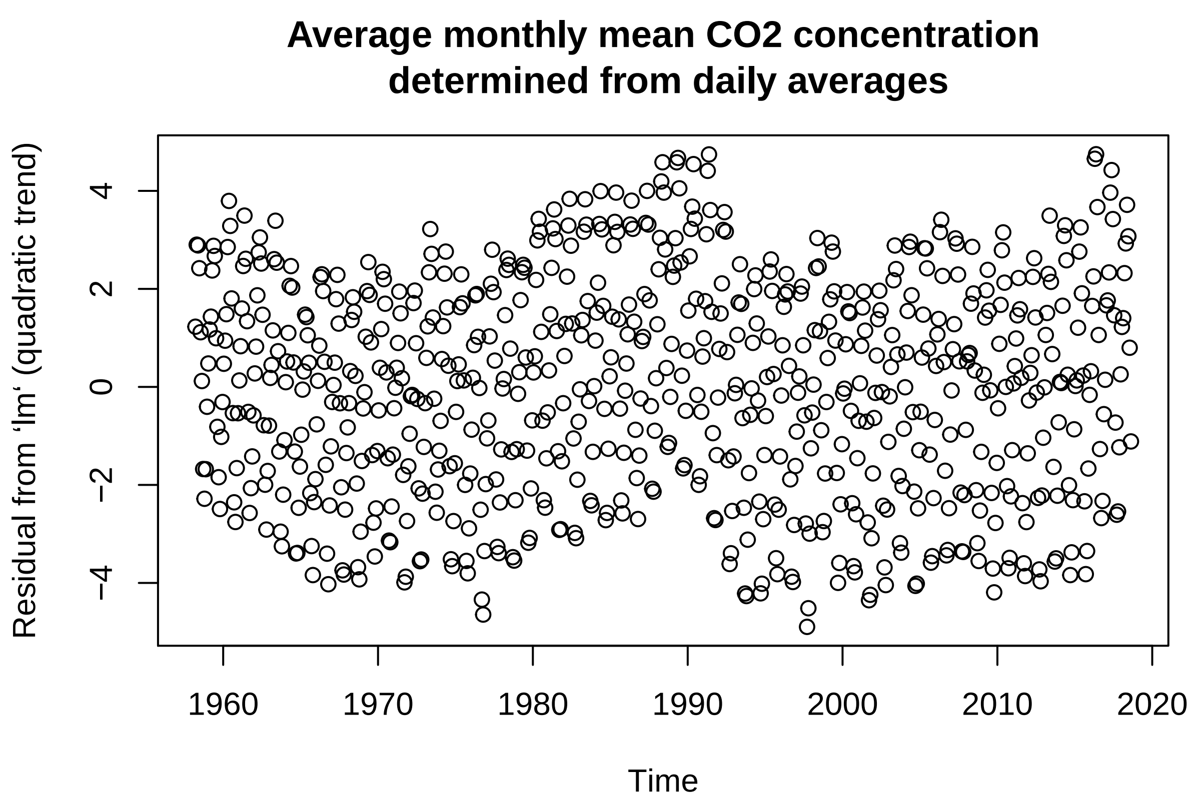
```
# Because of comment.char=#, all the first lines are skipped But we lose the
# header
co2 <- read.table(file = "ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_mm_mlo.txt",
    na.strings = "-99.99", stringsAsFactors = FALSE, col.names = c("year", "month",
        "time", "average", "interpolated", "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))
```



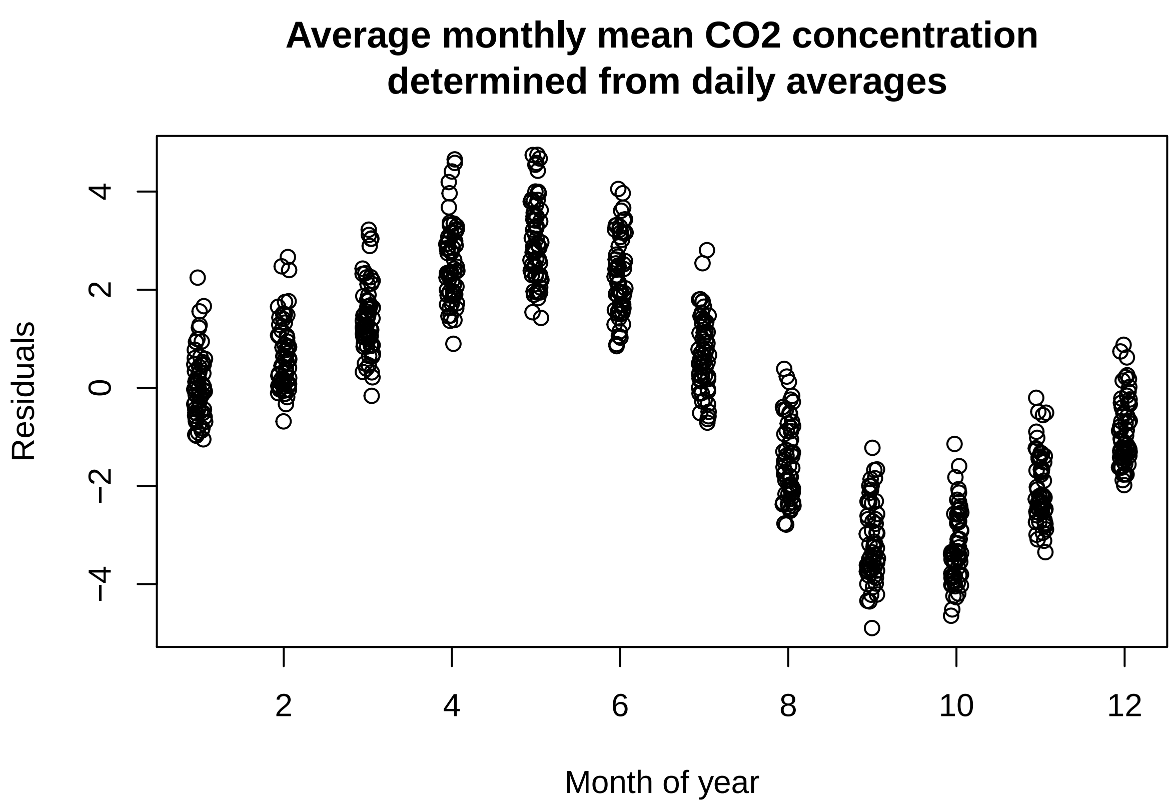

```
plot(co2$time, residuals(lin_mod), ylab = "Residual from `lm` (linear trend)",
    xlab = "Time", main = mcap)
```



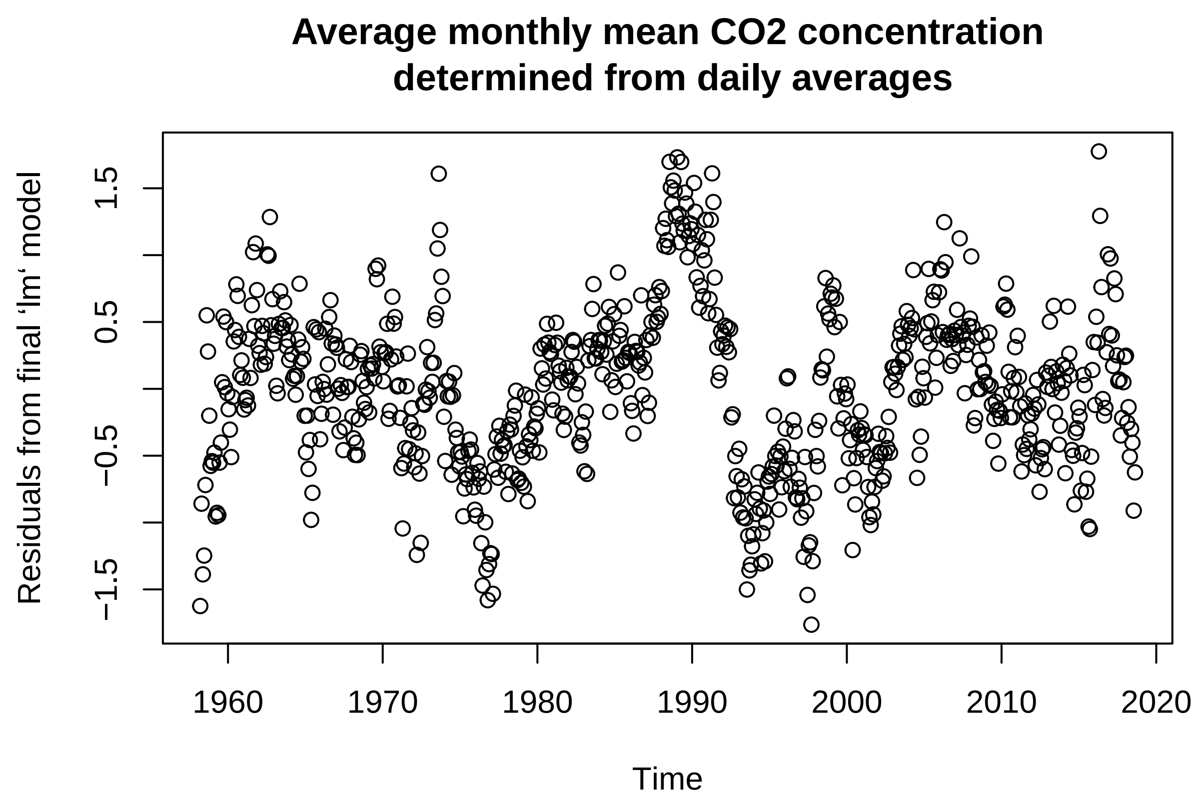

```
lin_mod <- update(lin_mod, . ~ . + I(time^2))
# same as lm(data=co2, interpolated ~ poly(time, degree = 2))
plot(co2$time, residuals(lin_mod), main = mcap, ylab = "Residual from `lm` (quadratic trend)",
     xlab = "Time")
```



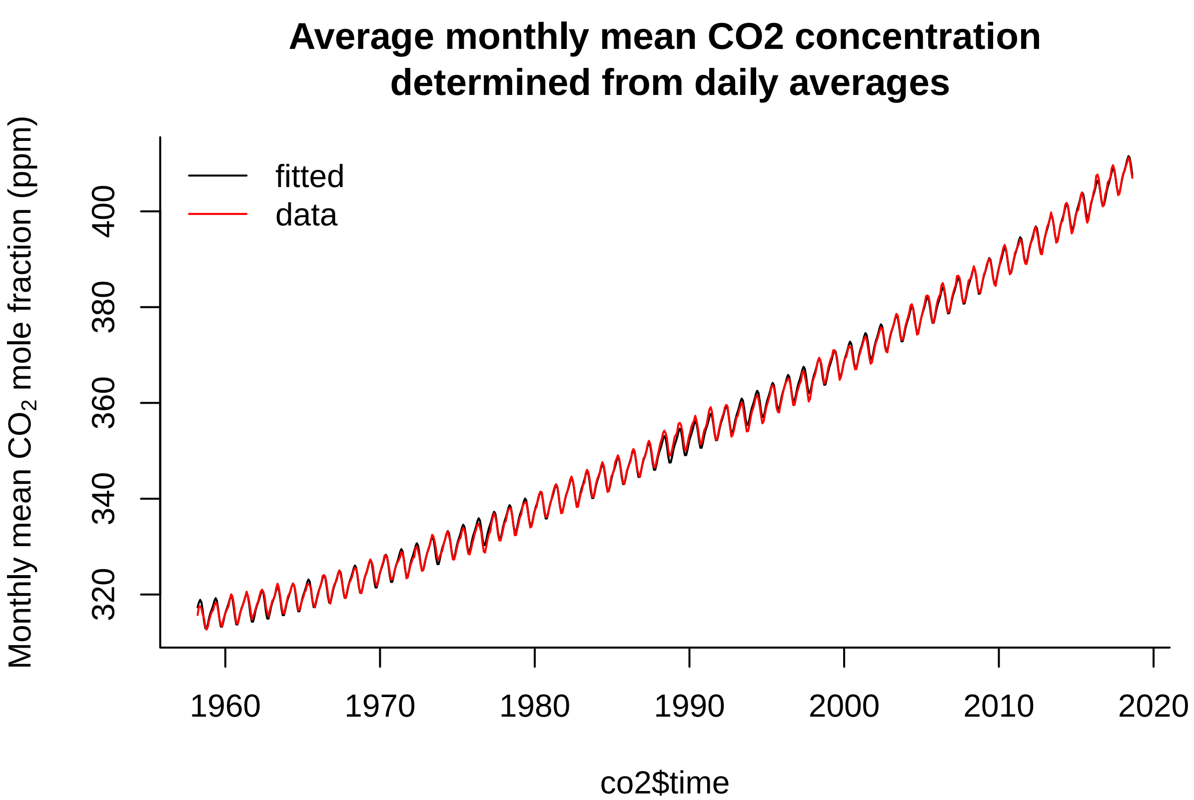
```
# Cast the full time series into a ts object
co2_ts <- with(co2, ts(data = interpolated, start = c(year[1], month[1]), end = c(tail(year,
    1), tail(month, 1)), frequency = 12))
with(co2, plot(jitter(month, 1/3), residuals(lin_mod), ylab = "Residuals", xlab = "Month of year",
    main = mcap))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



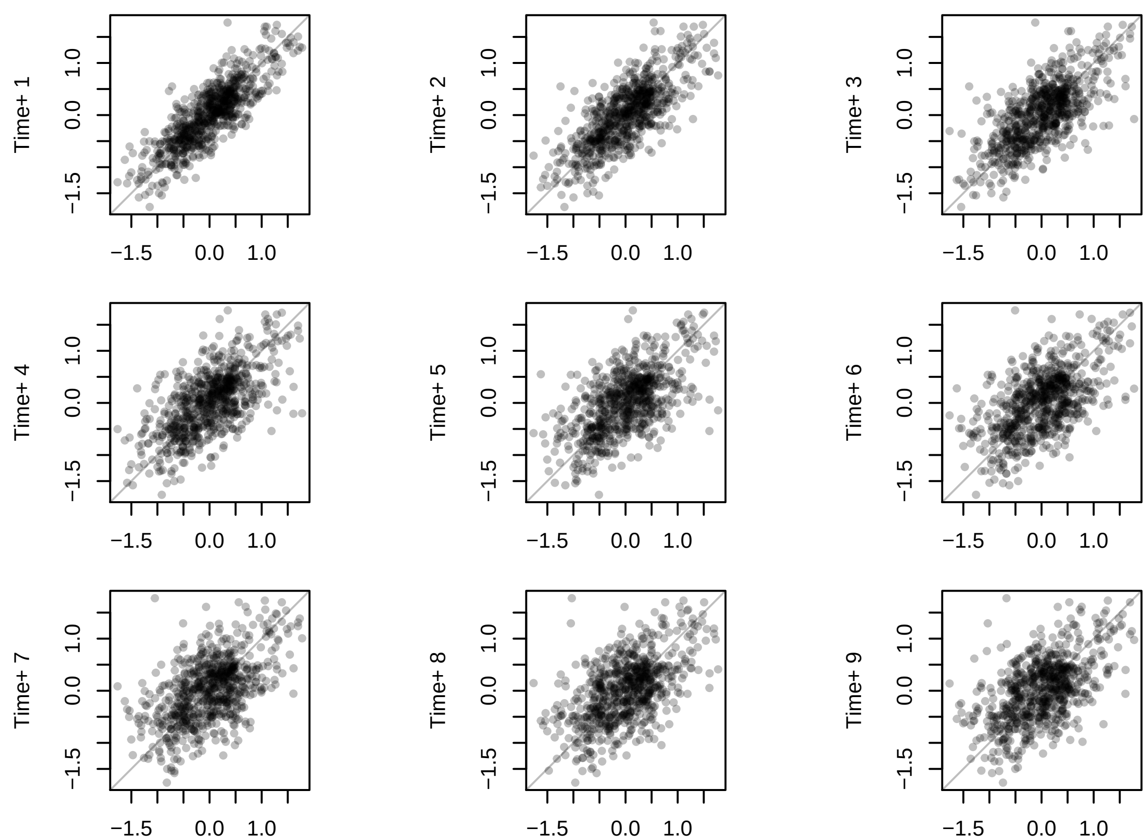

```
# Could create the basis manually
f_bs <- with(co2, fda::fourier(month, nbasis = 4, period = 12))[, -1]
lin_mod <- with(co2, lm(interpolated ~ splines::bs(time, df = 5, degree = 3) +
    f_bs))
# summary(lin_mod) Is there structure left in the residuals?
plot(co2$time, residuals(lin_mod), ylab =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



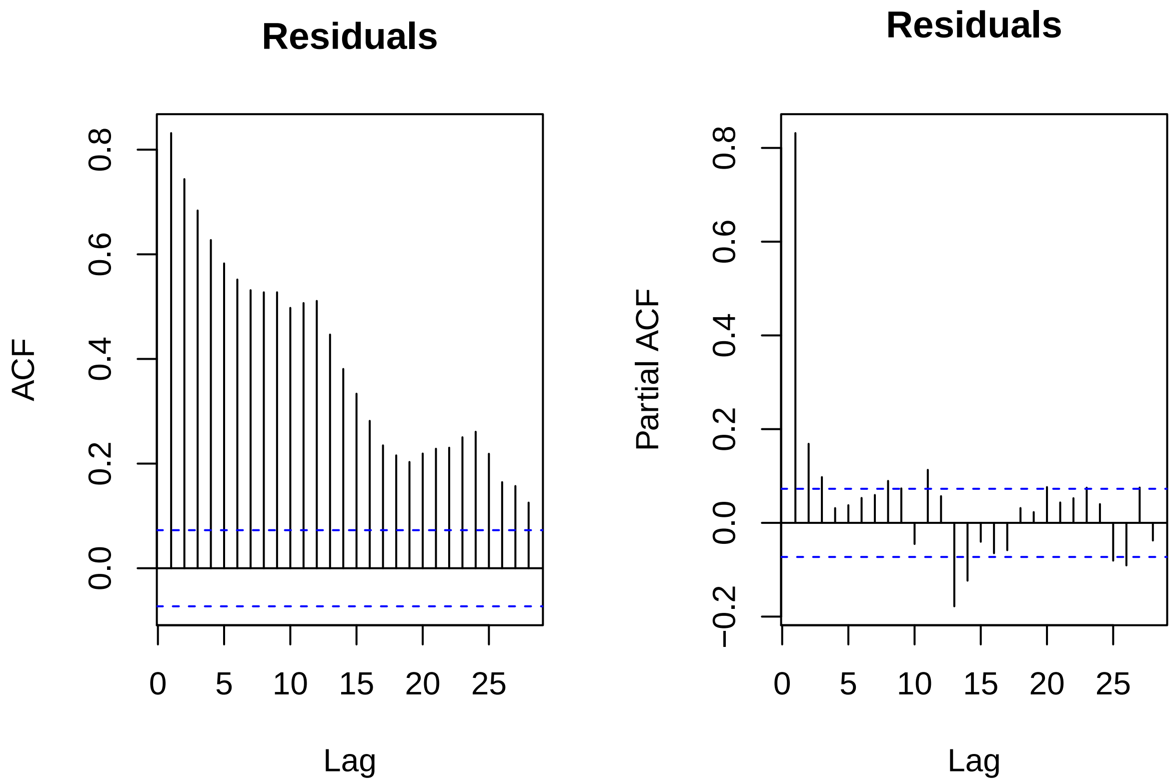

```
plot(co2$time, fitted(lin_mod), type = "l", ylab = ycap, main = mcap, bty = "l")
with(co2, lines(time, interpolated, col = 2))
legend(x = "topleft", legend = c("fitted", "data"), col = c(1, 2), lty = c(1,
    1), bty = "n")
```



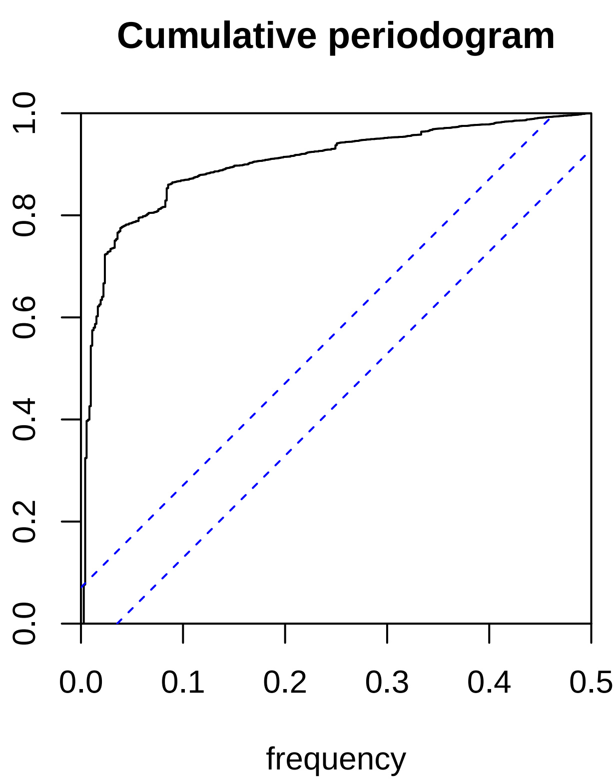
```r
# Trend not quite adequate because more exponential growth.  The trend does
# poorly in low-high observations Some discrepancy between the frequencies
# and the fitted Creates residual harmonic patterns - because trend minus
# fitted
res <- residuals(lin_mod)

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)
```



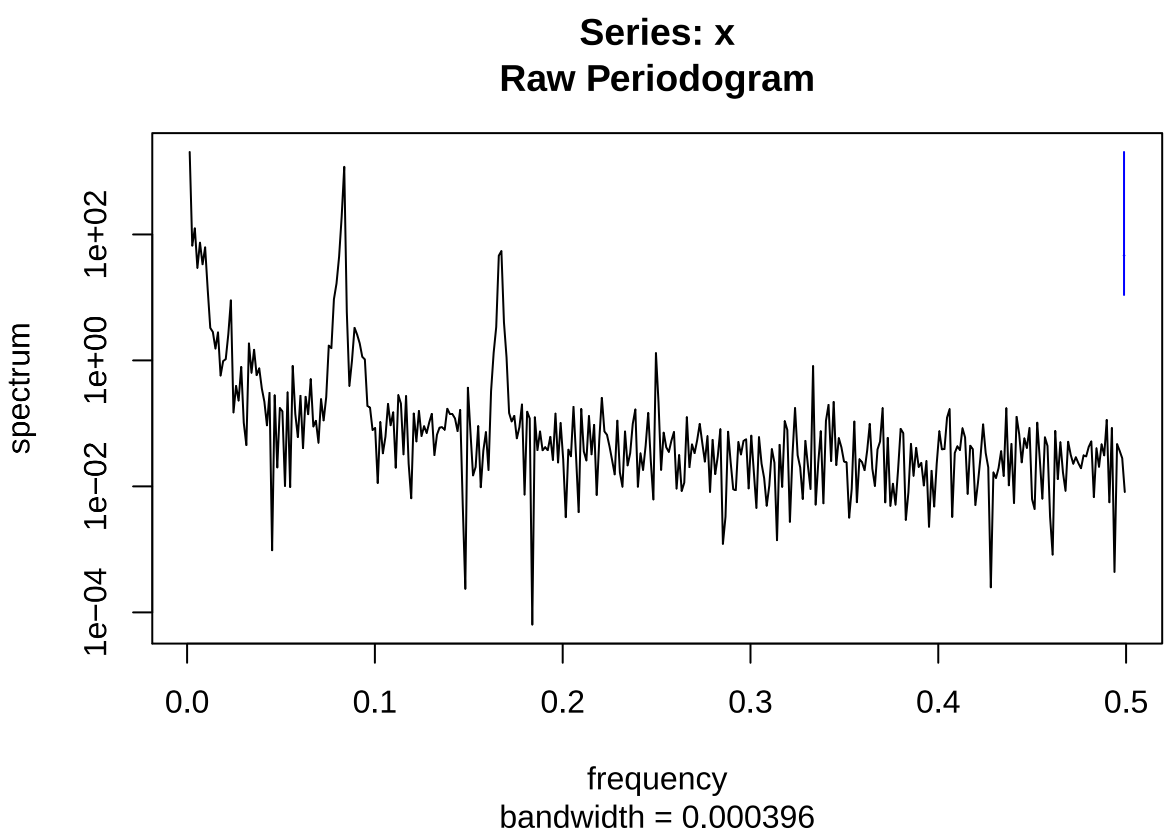

```r
par(mfrow = c(1, 2))
TSA::acf(res, main = "Residuals")
pacf(res, main = "Residuals")
```



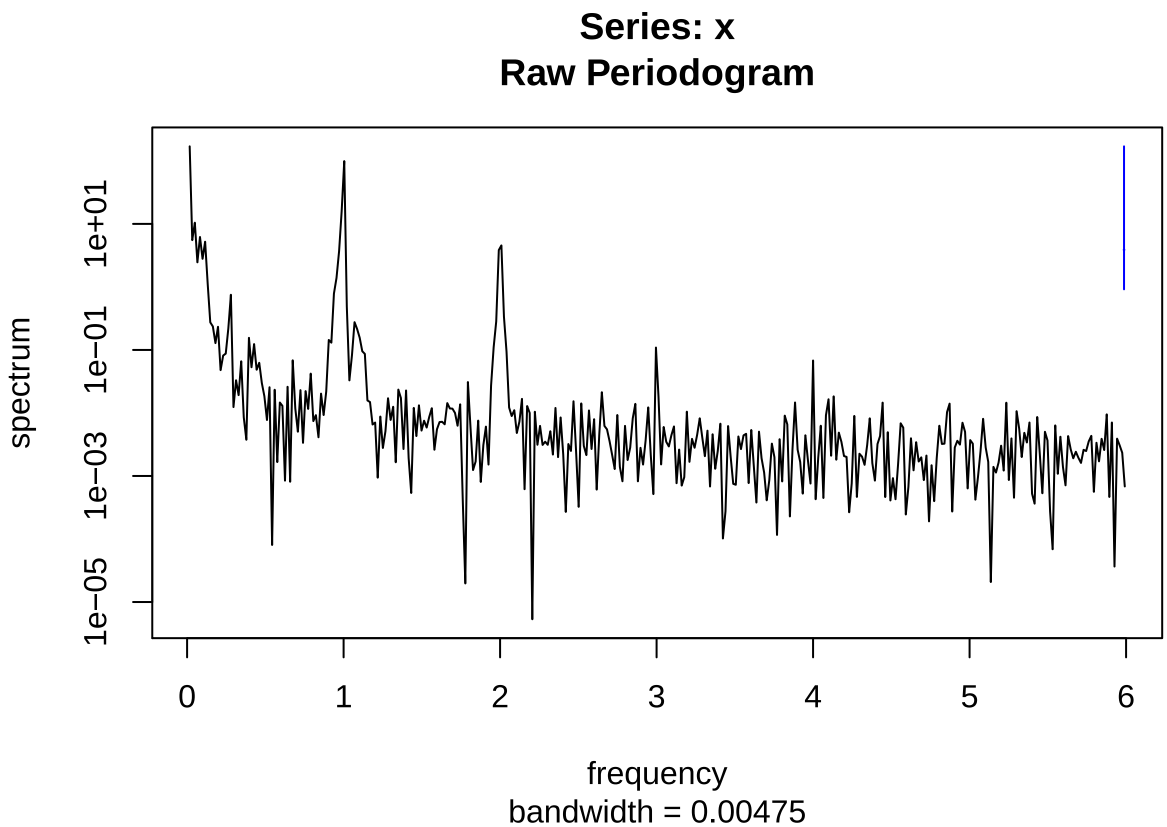

```
par(mfrow = c(1, 1))
# KS test: are residuals white noise?
cpgram(res, main = "Cumulative periodogram")
```



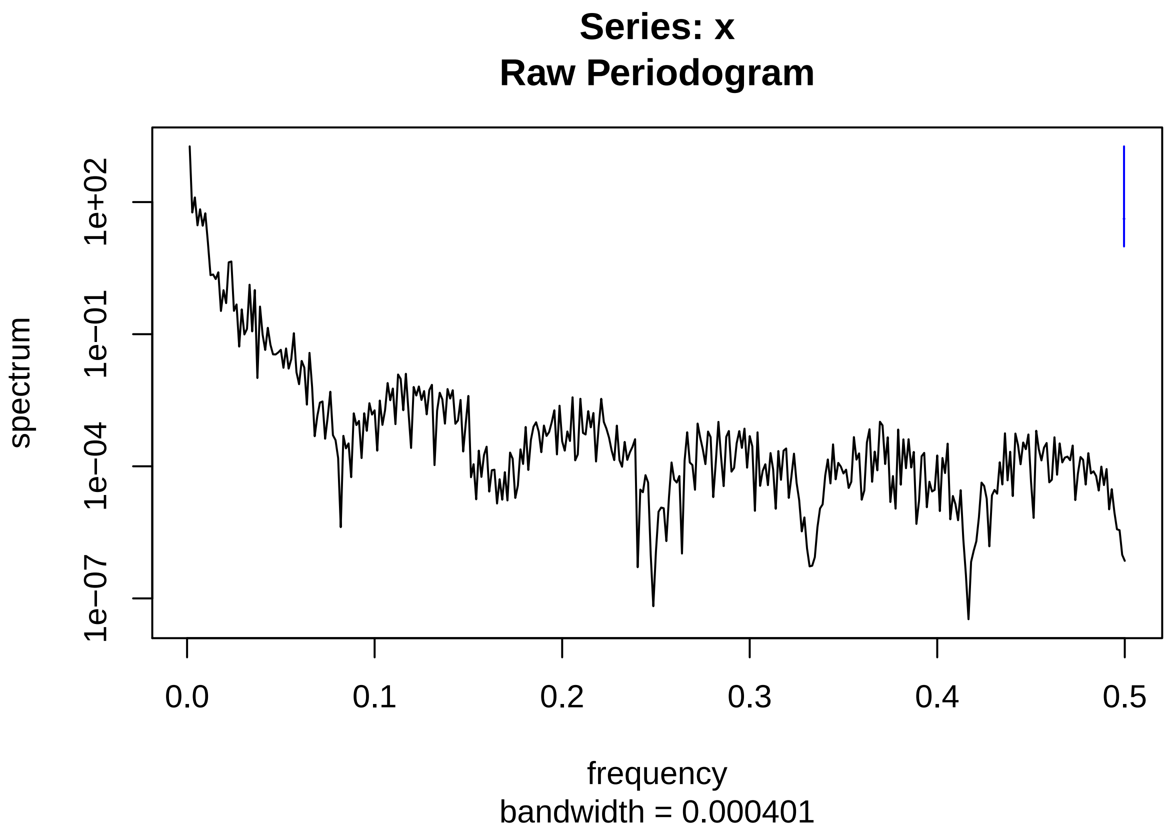
```r
# No, as one would expect

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



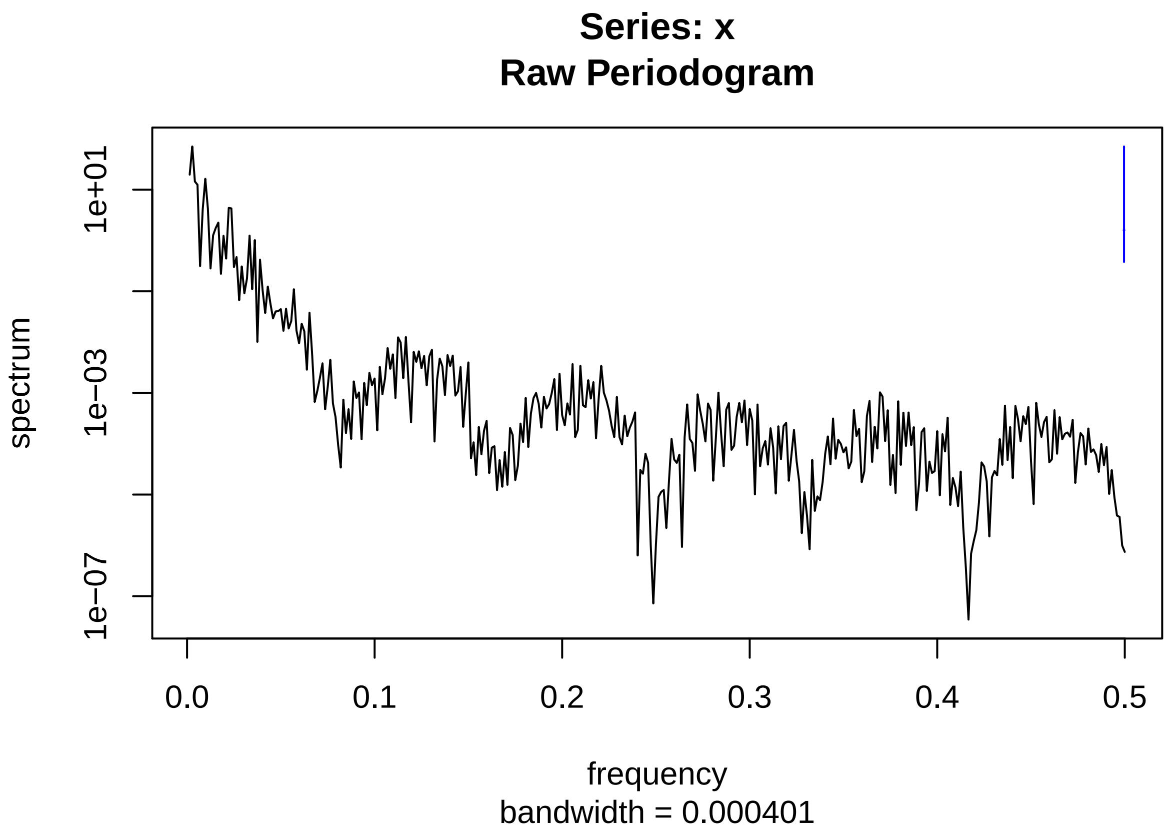

```r
# default with vector is to have frequency on [0,0.5]
spectrum(co2_ts)  #otherwise corresponds to frequency of `ts`, here yearly
```




```
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))))
```




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

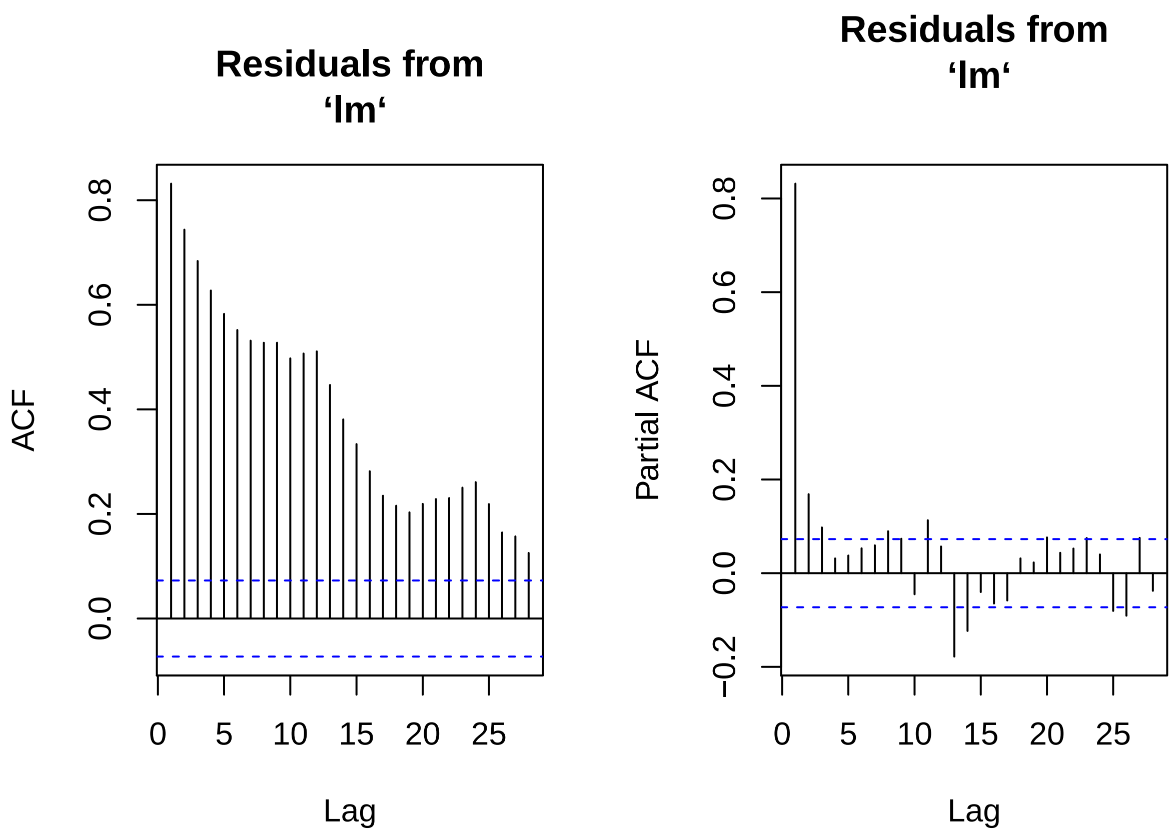
```

    KPSS Test for Level Stationarity

data:  res
KPSS Level = 0.17017, Truncation lag parameter = 6, p-value = 0.1
```

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

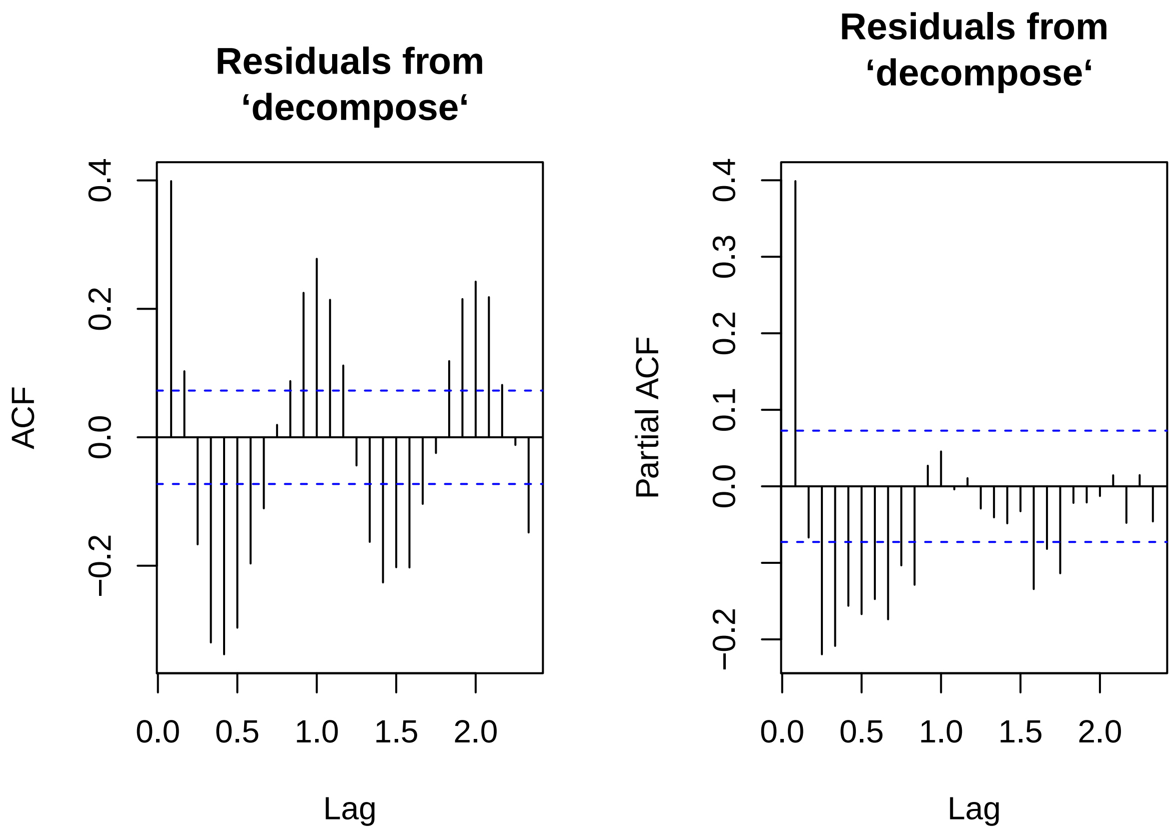
```

    KPSS Test for Trend Stationarity

data:  res
KPSS Trend = 0.17017, Truncation lag parameter = 6, p-value =
0.02986
```

```r
# 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`")
```

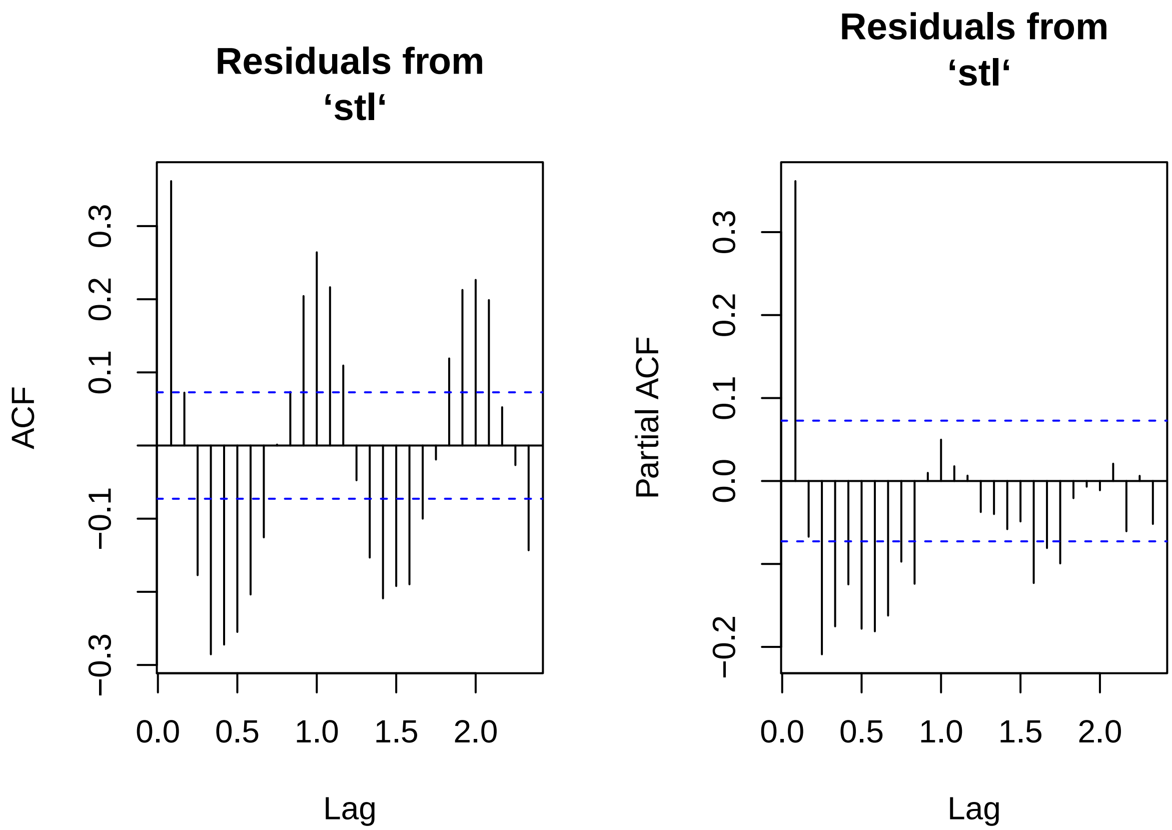
```r
pacf(res, na.action = na.pass, main = "Residuals from\n `lm`")
```




```r
# 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`")
```




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


Residuals from
'stl'
ACF
Lag
Residuals from
'stl'
Partial ACF
Lag

Chapter 3

Practical series 2

3.1 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)
```

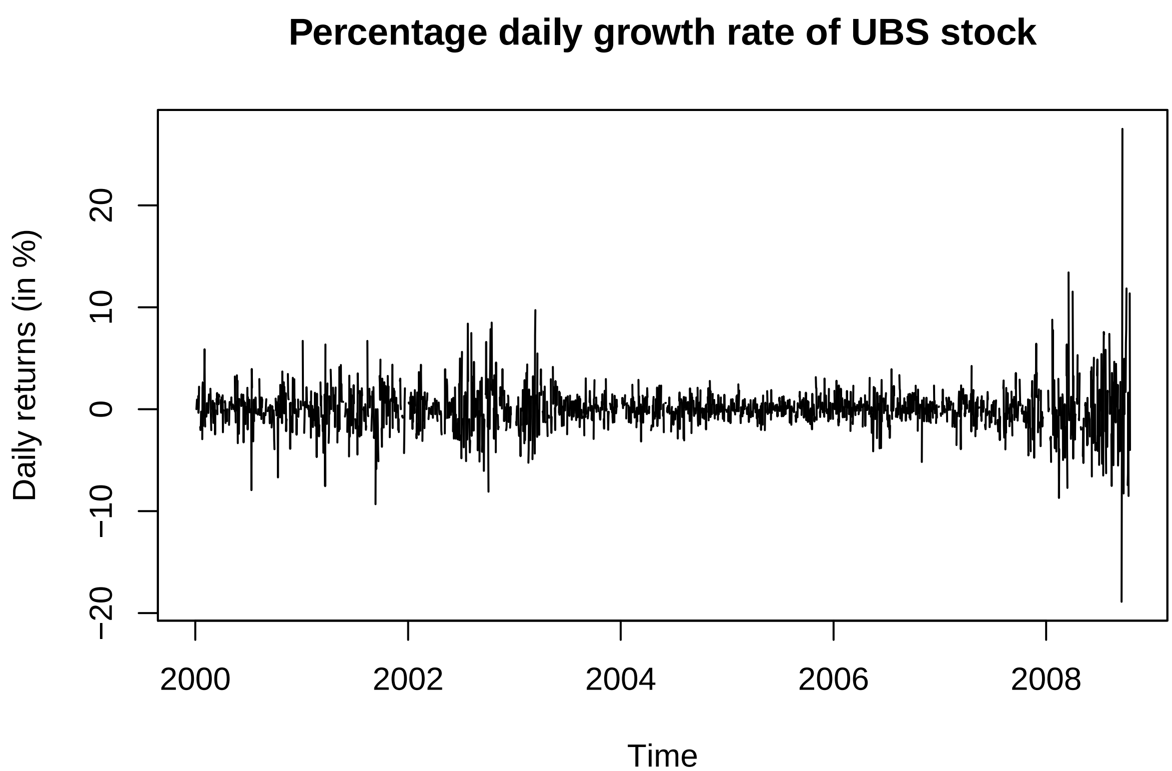
```
    Date UBS_OPEN UBS_HIGH UBS_LOW UBS_LAST UBS_VOLUME CSG_OPEN CSG_HIGH
1 1/1/00       NA       NA      NA       NA         NA       NA       NA
2 1/2/00       NA       NA      NA       NA         NA       NA       NA
3 1/3/00       NA       NA      NA       NA         NA       NA       NA
4 1/4/00    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)))
```

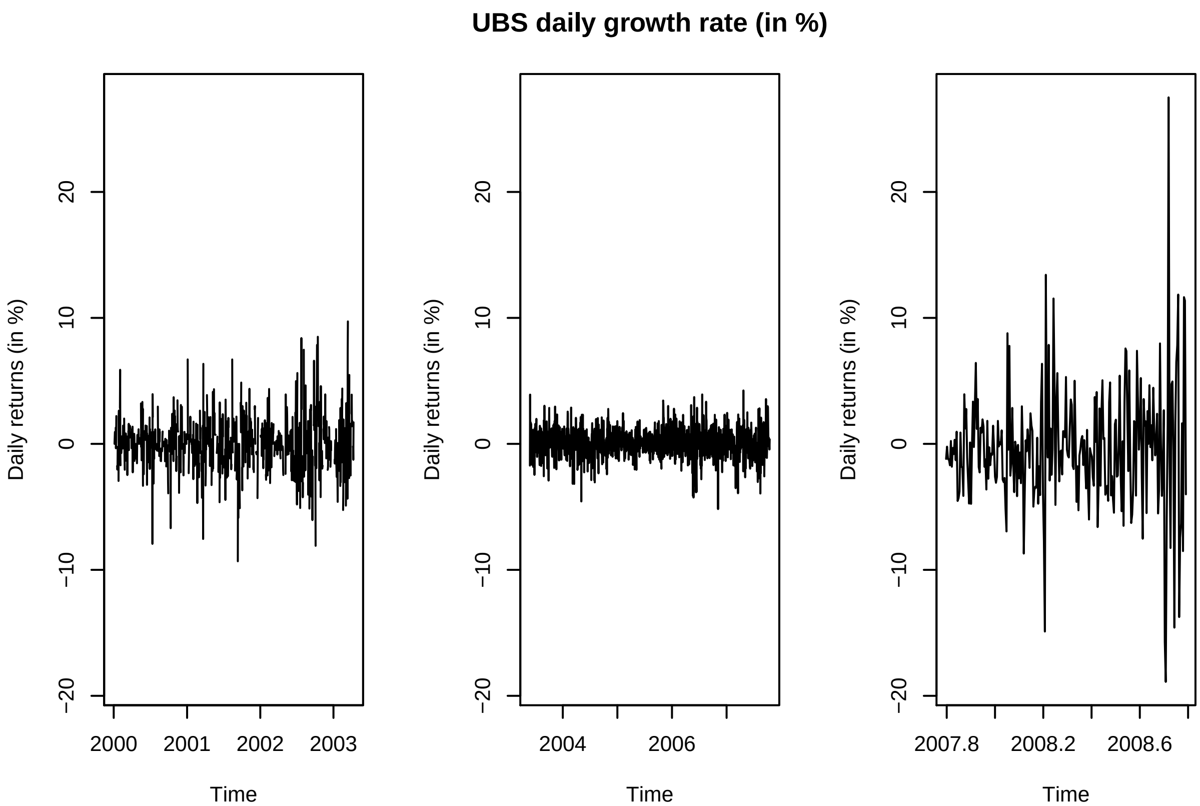
Objects of class `ts` store the dates from the vector start with observations as $(i-1)/\omega$. Thus, we specified in the above a vector encoded as 2000, 2000+1/365.25, …This means that missing values are not handled. In contrast, `xts` objects keep the time stamps from a `Date` object. The function `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
```

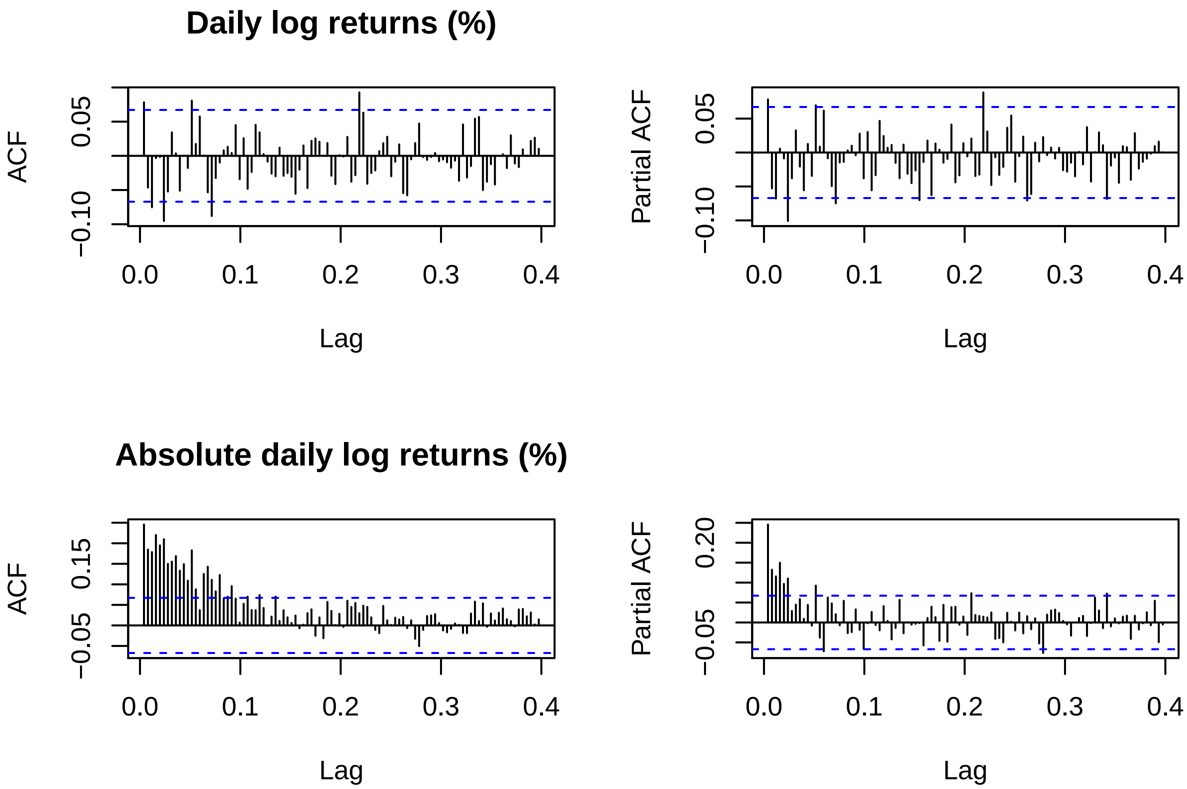



```
# compute log returns
UBS.ret <- 100 * diff(log(UBS))

# 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)))
```