# Computer exercises 3

## Demo exercises

### 3.1

Study the following time series.

| File | Variable | Description | Interval | Length |
|---|---|---|---|---|
| intel.txt | Intel_Close | Intel stock price | Exchange day | $n = 20$ |
| | Intel_Volume | Intel stock volume | | |
| sunspot.txt | Spots | Number of sunspots | 1 year | $n = 215$ |
| mlco2.txt | MLCO2 | Carbon dioxide measurements from the Mauna Loa volcano | 1 month | $n = 216$ |
| sales.txt | Sales | Sales volume of a wholesaler | 1 month | $n = 144$ |
| passengers.txt | Passengers | Number of airline passengers on international routes in USA | 1 month | $n = 144$ |

### Solution

First we read all the data.

```r
# Read the data
intel <- read.table("data/intel.txt", header = TRUE)
sunspot <- read.table("data/sunspot.txt", header = TRUE)
mlco2 <- read.table("data/mlco2.txt", header = TRUE, row.names = 1)
sales <- read.table("data/sales.txt", header = TRUE, row.names = 4)
passengers <- read.table("data/passengers.txt", header = TRUE, row.names = 4)

str(intel)
```

```
## 'data.frame':    20 obs. of  3 variables:
##  $ Date        : int  950801 950802 950803 950804 950807 950808 950809 950810 950811 950814 ...
##  $ Intel_Close : num  65 65 62.8 63 63.9 ...
##  $ Intel_Volume: num  11242 16690 14613 8009 6442 ...
```

```r
str(sunspot)
```

```
## 'data.frame':    215 obs. of  2 variables:
##  $ Year : int  1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 ...
##  $ Spots: num  80.9 83.4 47.7 47.8 30.7 12.2 9.6 10.2 32.4 47.6 ...
```

```r
str(mlco2)
```

```
## 'data.frame':    216 obs. of  1 variable:
##  $ MLCO2: num  14.9 15.6 16.3 17.6 17.9 ...
```

```r
str(sales)
```

```
## 'data.frame':    144 obs. of  3 variables:
##  $ Year : int  1970 1970 1970 1970 1970 1970 1970 1970 1970 1970 ...
##  $ Month: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Sales: int  129 122 137 141 145 144 143 140 140 148 ...
```

```
str(passengers)
```

```
## 'data.frame':    144 obs. of  3 variables:
##  $ Year      : int  1949 1949 1949 1949 1949 1949 1949 1949 1949 1949 ...
##  $ Month     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Passengers: int  112 118 132 129 121 135 148 148 136 119 ...
```

When working with time series data, data frames and vectors are not always the best structures for storing data. Instead, there are special classes designed for time series data such as the following ones, among many others.

- `ts` – base R version.
- `xts` – Support for `date` and `datetime` (`POSIXct`) indices among other features.
- `tsibble` – For working with time series data in tidyverse.

During the course we focus mainly on solutions provided by base `R`. Thus we will work with `ts` objects. Next we create `ts` objects from the original data.

```
# Vectors, matrices and data frames can be given as an input. In the case of
# matrices and data frames, It is assumed that each column represents one
# univariate time series.
intel_ts <- ts(intel[, -1])

# One can specify start of the time series
sunspot_ts <- ts(sunspot$Spots, start = 1749)

# One can also specify number of observation per unit of time. In below cases
# we say that there are 12 observations per year, i.e., one per month.
mlco2_ts <- ts(mlco2$MLCO2, frequency = 12)
sales_ts <- ts(sales$Sales, start = 1970, frequency = 12)

# Instead of specifying frequency one can set the fraction of the sampling
# period between successive observations with argument deltat.
passengers_ts <- ts(passengers$Passengers, start = 1949, deltat = 1 / 12)
```

Notice how the `plot` function works differently for data frames and vectors compared to `ts` objects. You can compare results of Figures 1 and 2. However, you can always invoke the plot method of `ts` class by calling `plot.ts`.

```
# Without setting yax.flip = TRUE y-tick labels are on top of each other.
plot(intel_ts, yax.flip = TRUE)
```

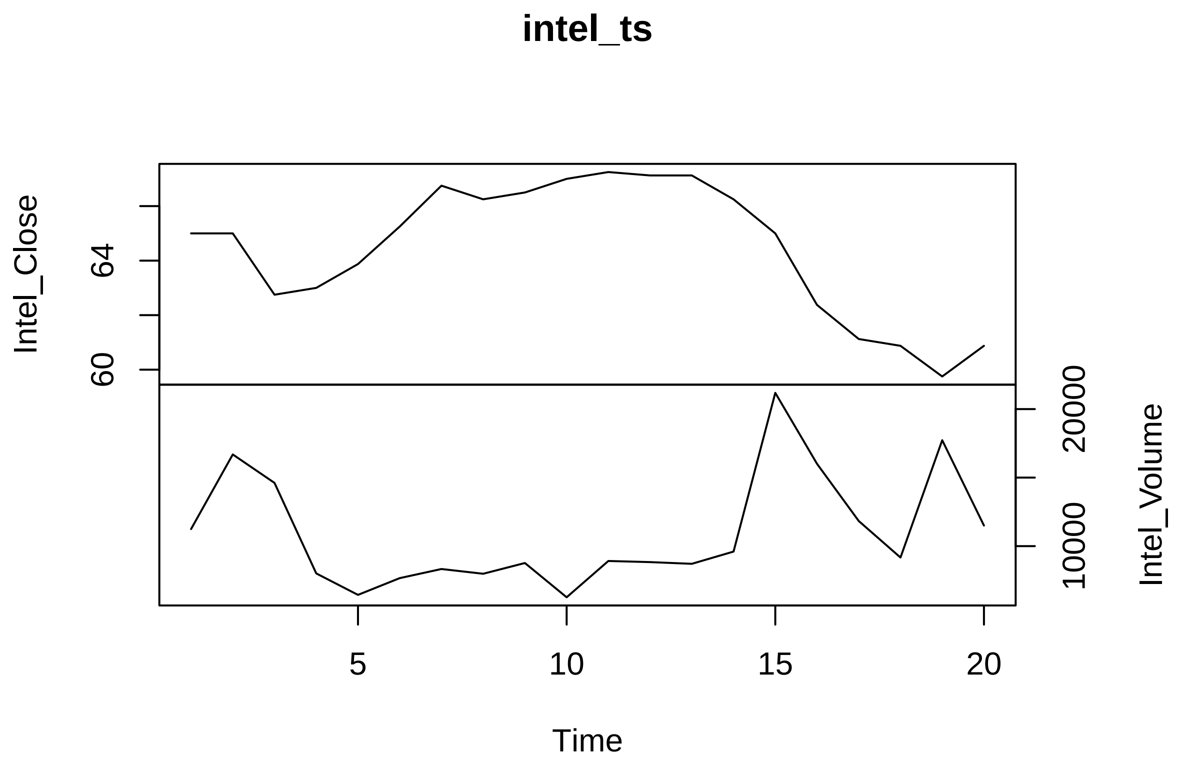

Figure 1: Plots of time series `Intel_Close` and `Intel_Volume`, respectively.

```
plot(intel[, -1])
```

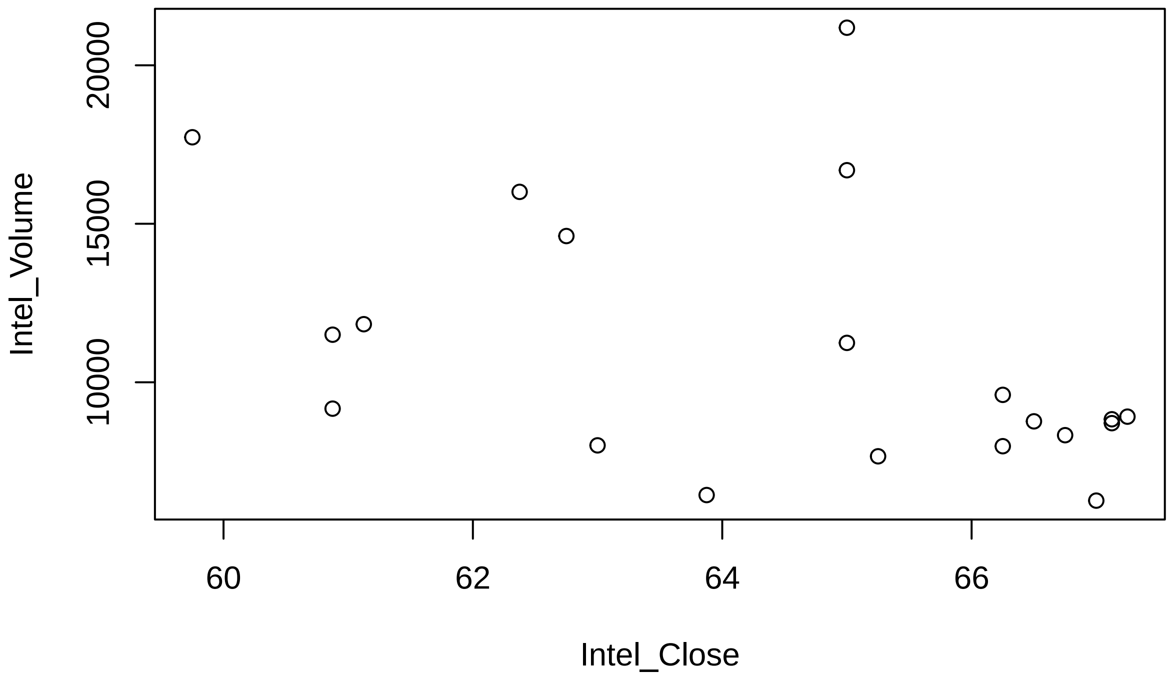

Figure 2: Scatter plot of variables `Intel_Close` and `Intel_Volume`.

**Intel**

Variable `Intel_Close` represents the Intel stock price in New York stock exchange at the end of the trading day. The time span is four weeks. From the upper panel of Figure 1 we can deduce the following.

- Trend:
  - No clear trend but the level of the time series alters.
- Seasonality:
  - No seasonality.
- Stationarity:
  - Due to the small number of observations, it is hard to say anything about stationarity. However, the mean of the time series does not seem to be constant, which indicates that the time series might not be stationary. This time series is considered with more detail in Problem 3.3.

Variable `Intel_Volume` represents the daily volume of Intel stocks traded in New York stock exchange. The time span is four weeks. From the lower panel of Figure 1 we can deduce the following.

- Trend:
  - No clear trend but the level of the time series alters.
- Seasonality:
  - No seasonality.

- Stationarity:
  - Due to the small number of observations, it is hard to say anything about stationarity. However, the mean of the time series does not seem to be constant, which indicates that the time series might not be stationary.

**Sunspots**

Variable `Spots` represents number of yearly sunspots. From Figure 3 we can deduce the following.

```
plot(sunspot_ts)
```

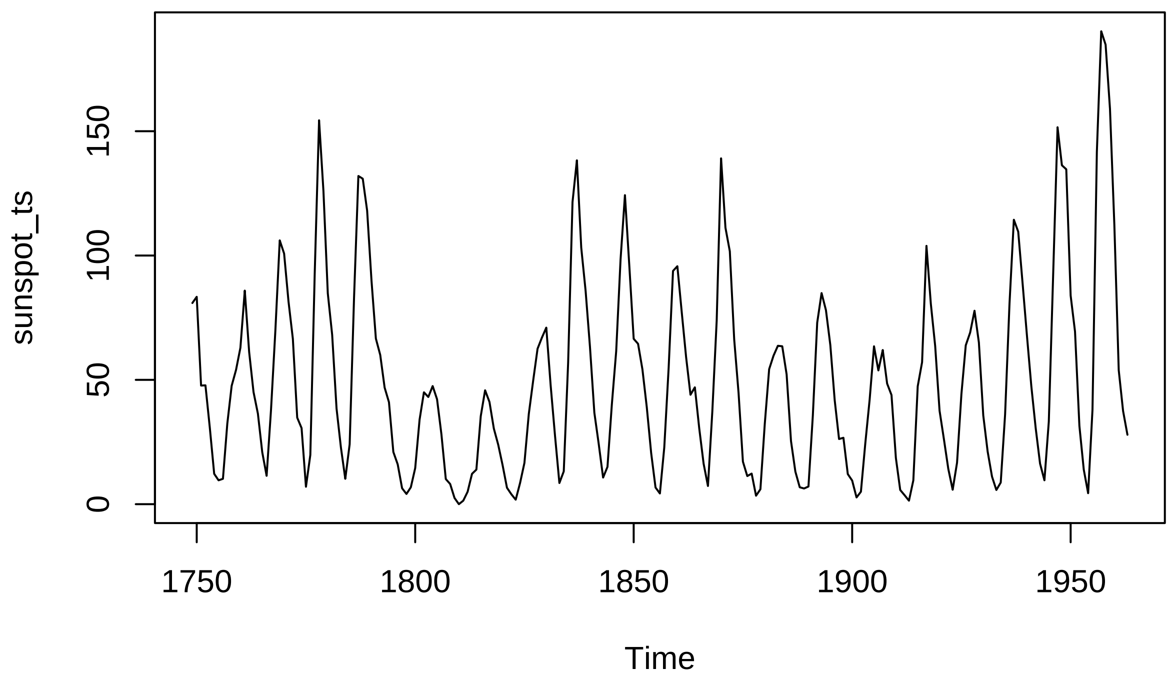

Figure 3: Plot of time series `Spots`.

- Trend:
  - No trend.
- Seasonality:
  - Seasonal component with period that is approximately 11 years. The amplitude of the time series alters.
- Stationarity:
  - Time series does not seem to be stationary, since it clearly has a seasonal component.

**Mlco2**

Variable `MLCO2` represents carbon dioxide measurements of the Mauna Loa volcano in Hawaii. From Figure 4 we can deduce the following.

```
plot(mlco2_ts)
```

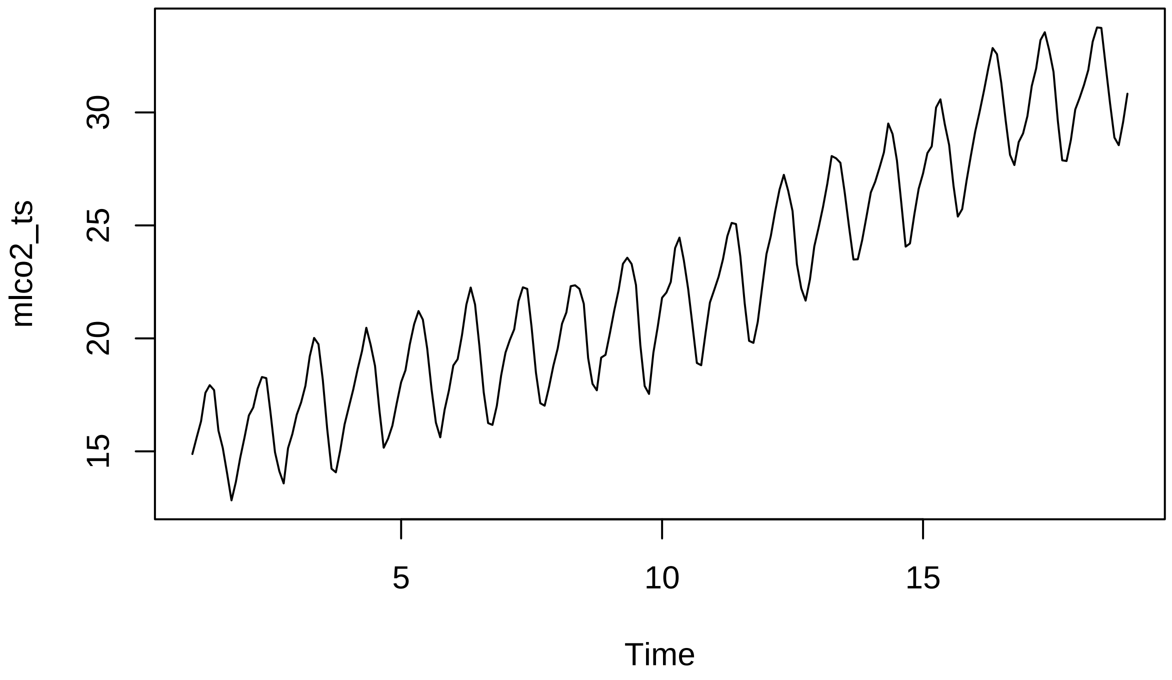

Figure 4: Plot of time series `MLCO2`.

- Trend:
  - Linear upward trend.
- Seasonality:
  - Quite regular seasonality with a period of 12 months. Amplitude of the seasonal component stays constant.
- Stationarity:
  - The time series does not seem to be stationary, since it has an upward trend and a seasonal component.

**Sales**

Variable `Sales` represents sales volume of a wholesaler per month. From Figure 5 we can deduce the following.

```
plot(sales_ts)
```

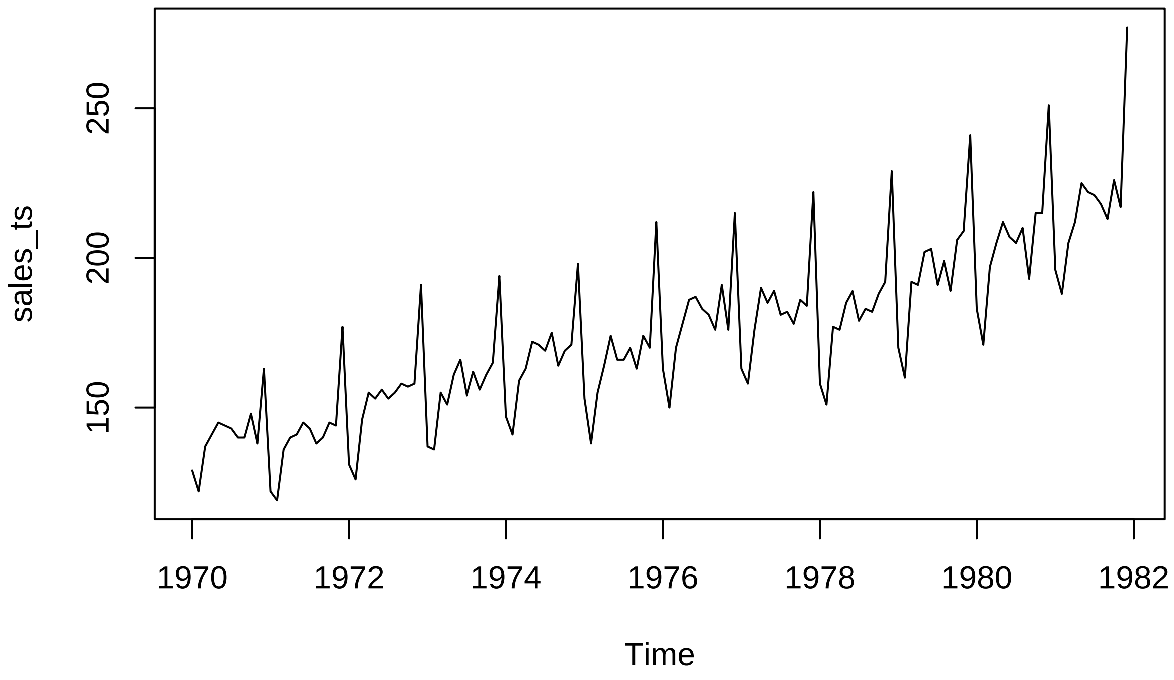

Figure 5: Plot of time series `Sales`.

- Trend:
  - Upward trend.
- Seasonality:
  - Quite regular seasonality with a period of 12 months. The amplitude of the seasonal component increases with the level of the time series.
- Stationarity:
  - The time series does not seem to be stationary, since it has an upward trend and a seasonal component.

**Passengers**

Variable `Passengers` represents number of airline passengers on international routes in the USA. From Figure 6 we can deduce the following.

```
plot(passengers_ts)
```

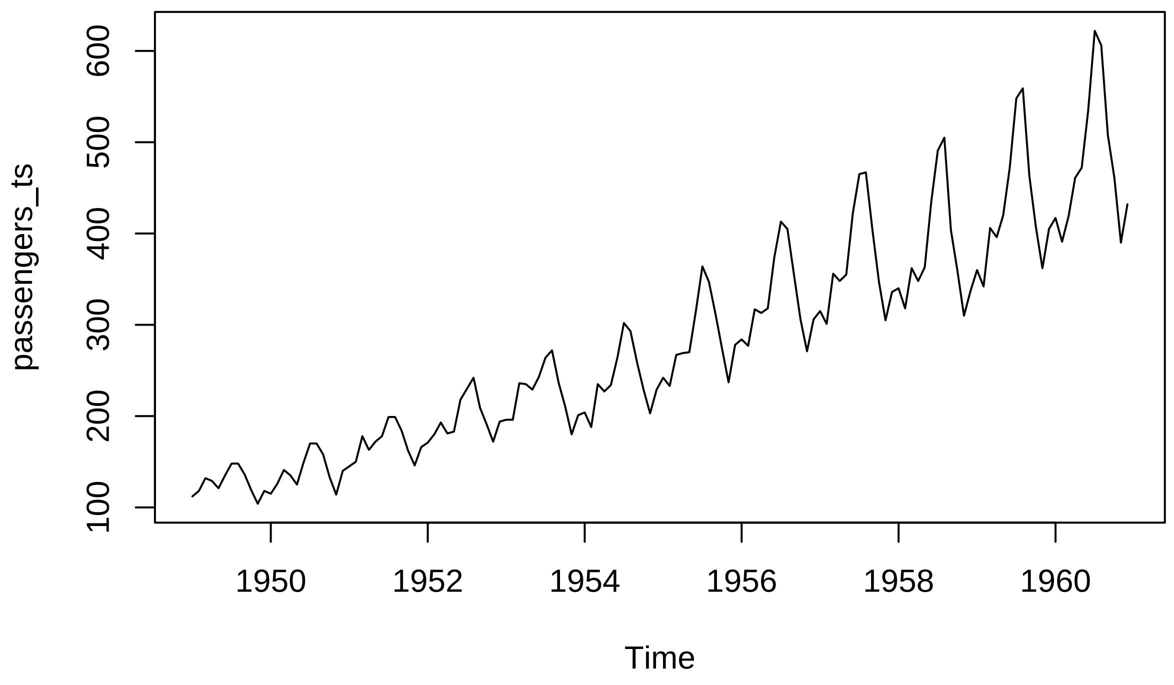

Figure 6: Plot of time series `Passengers`.

- Trend:
  - Upward slightly curvilinear trend.
- Seasonality:
  - Quite regular seasonality with a period of 12 months. The amplitude of the seasonal component increases with the level of the time series.
- Stationarity:
  - The time series does not seem to be stationary, since it has an upward trend and a seasonal component.

### 3.2

The file `passengers.txt` contains a time series named `Passengers`. Plot the time series by using both linear and logarithmic scales on the $y$-axis. Compare the plots.

| File | Variable | Description | Interval | Length |
|---|---|---|---|---|
| `passengers.txt` | `Passengers` | Number of airline passengers on international routes in USA | 1 month | $n = 144$ |

## Solution

Figure 7 illustrates, that the amplitude of the seasonal component of the original time series increases together with the level of the time series (left figure). When logarithmic scale is used, the amplitude is almost constant (right figure). On the other hand, the (slight) curvature of the trend of the original time series gets (slightly) overcompensated with logarithmic scale. Indeed, logarithmic transform is often used to stabilize variance when standard deviation increases linearly with the level.

```r
par(mfrow = c(1, 2), mar = c(2.5, 2.5, 1.5, 1.5))
plot(passengers_ts, main = "Passengers")
plot(log(passengers_ts), main = "Log(Passengers)")
```

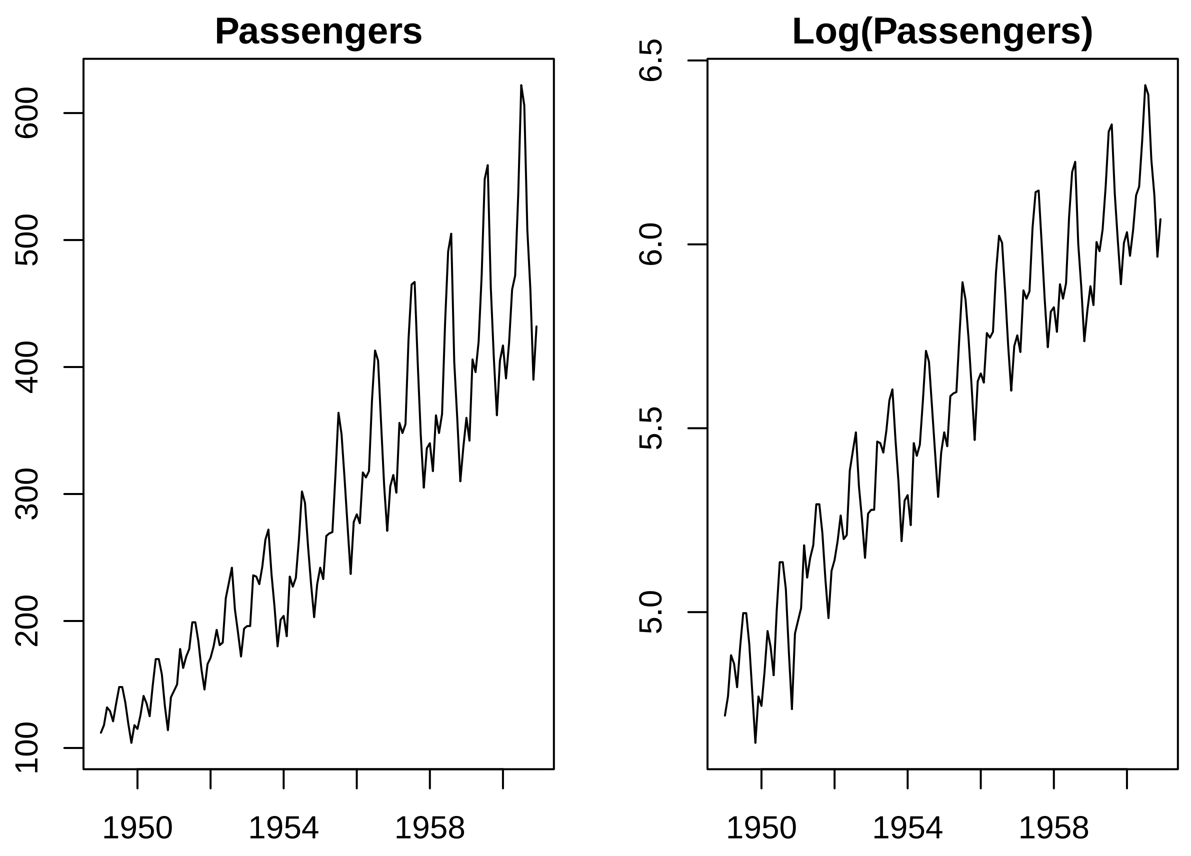

Figure 7: Plot of `Passengers` time series with linear and logarithmic scales.

## 3.3

Study the following time series by estimating their autocorrelation and partial autocorrelation functions.

| File | Variable | Description | Interval | Length |
|---|---|---|---|---|
| `intel.txt` | `Intel_Close` | Intel stock price | Exchange day | $n = 20$ |
| `sunspot.txt` | `Spots` | Number of sunspots | 1 year | $n = 215$ |

## Solution

The estimated *autocorrelation function* (ACF) and *partial autocorrelation function* (PACF) can be computed with functions `acf` and `pacf`, respectively. Argument `lag.max` controls for how many lags estimated ACF and PACF are calculated. Maximum number of lags for which estimates are computed is $n-1$ for a time series of length $n$. Thus by setting `lag.max = Inf` we get estimated ACF/PACF for all possible lags. However, with small sample size $n$ estimated autocorrelations for large lag $k$ can be unreliable, since estimates are based only on a few observations.

Note that while theoretical ACF and PACF are defined only for stationary time series, *estimated* ACF and PACF are also defined for non-stationary time series. This can be useful since trend and seasonality of non-stationary time series are often visible in estimated ACF and PACF. On the other hand, estimated ACF and PACF of stationary time series typically decay to zero rapidly.

The blue dashed lines in ACF/PACF plots (see ,e.g., Figure 8) indicate statistical significance with 5% level of significance assuming i.i.d. observations (see lecture slides 4 for details). That is, roughly speaking autocorrelations inside blue lines can be interpreted as zero (not significant).

### Intel_Close

Figure 8 shows estimated ACF and estimated PACF for time series `Intel_Close`.

```
par(mfrow = c(1, 2))
acf(intel_ts[, "Intel_Close"], lag.max = Inf, main = "ACF of Intel_Close")
pacf(intel_ts[, "Intel_Close"], lag.max = Inf, main = "PACF of Intel_Close")
```

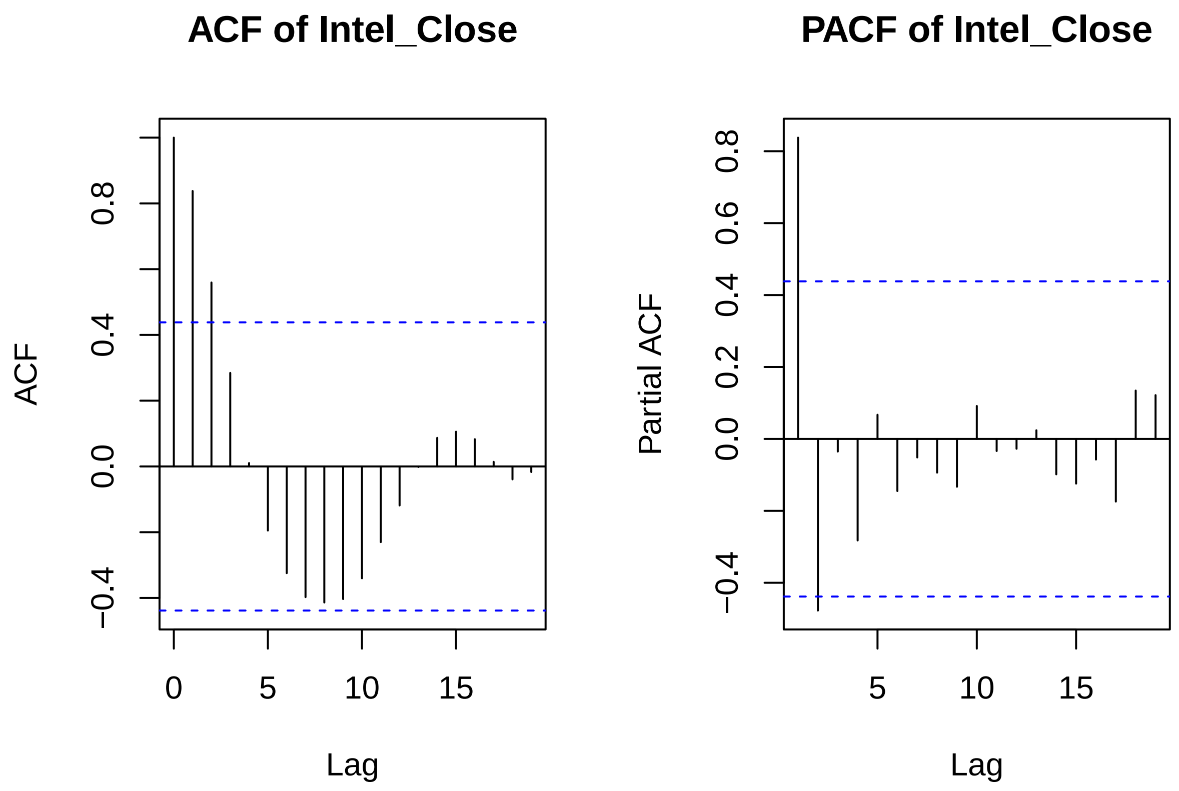

Figure 8: Estimated ACF and PACF of time series `Intel_Close`.

Time series `Intel_Close` could be stationary based on Figure 8 and hence, it does not require differencing. The level of the time series changes, but the behavior is calm locally. We cannot see a monotonic trend or a visible seasonal component.

#### Sunspots

Based on Exercise 3.1, time series `Spots` is not stationary. Next, we study the characteristics of estimated ACF and PACF of a non-stationary time series. Estimated ACF and PACF of time series `Spots` is showed in Figure 9.

```r
par(mfrow = c(1, 2))
acf(sunspot_ts, main = "ACF of Spots")
pacf(sunspot_ts, main = "PACF of Spots")
```

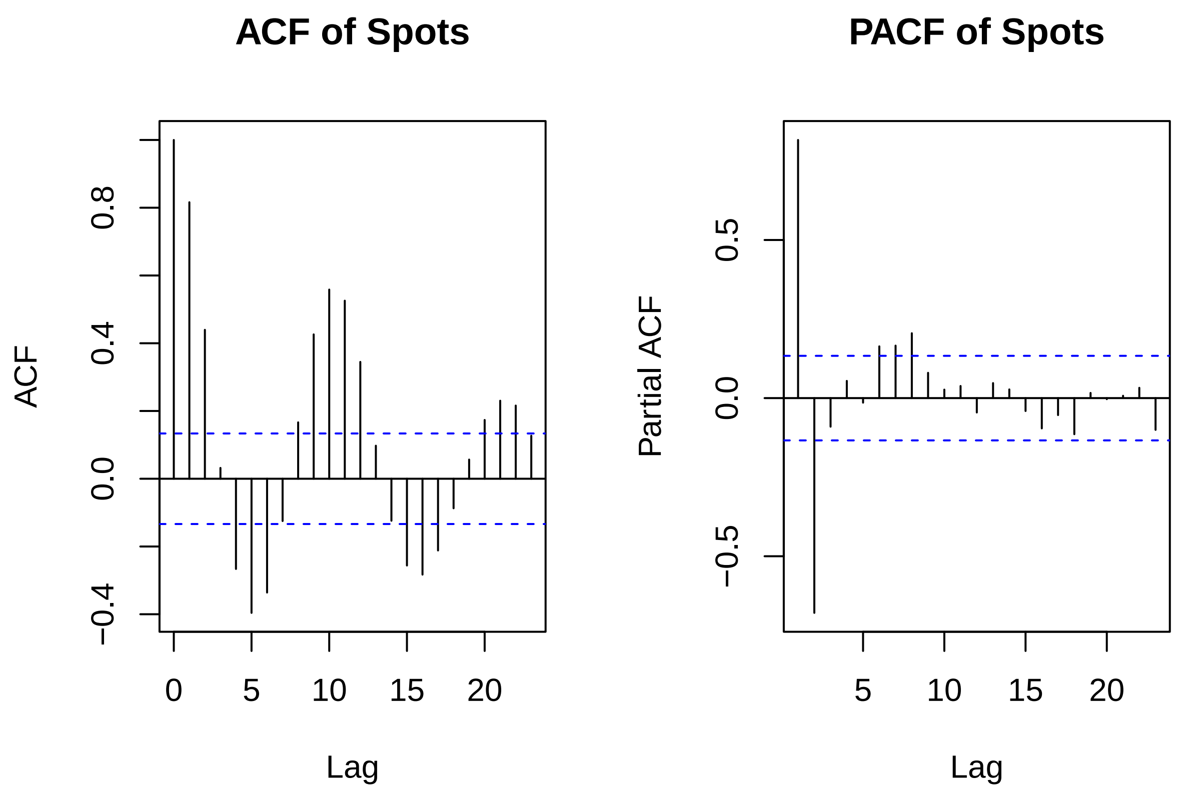

Figure 9: Estimated ACF and PACF of time series `Spots`.

Note that the seasonality is clearly visible in the autocorrelation function. By enlarging Figure 9, we can see that the period of the season seems to be about 11 years.

## Homework

### 3.4

Consider the time series `Sales` from the file `sales.txt`. Apply differencing, seasonal differencing and logarithmic transformations to remove the trend, the seasonality and the increasing variance. Which difference operations did you apply? Visualize both the original and the transformed time series. **Hint:** Difference operators are given by the function `diff` in R.

| File | Variable | Description | Interval | Length |
|---|---|---|---|---|
| `sales.txt` | `Sales` | Sales volume of a wholesaler | 1 month | $n = 144$ |