ASTA

The ASTA team

Contents

1 Concepts, terminology and examples **2**

1.1 Plotting, manipulation, trends and seasonality 2

1.2 Stochastic trend 4

1.3 Multiple series 5

1.4 Decomposition 7

2 Stationarity and autocorrelation **11**

2.1 Correlogram (empirical acf) 14

2.2 Typical patterns in correlograms 16

2.3 Partial autocorrelation function 20

3 Basic stochastic models **21**

3.1 White noise 21

3.2 Random walk 23

4 Auto-regressive (AR) models **27**

4.1 Auto-regressive model of order 1: AR(1) 27

4.2 Auto-regressive models of higher order 34

5 Moving average models **37**

5.1 Simulation of MA(q) processes 38

6 Mixed models: Auto-regressive moving average models **40**

7 Continuous time stochastic processes **42**

7.1 Data example 42

8 Wiener process **42**

9 Differential equation models **43**

9.1 Ordinary differential equations 43

9.2 Stochastic differential equations 44

9.3 Fitting SDE models to data 49

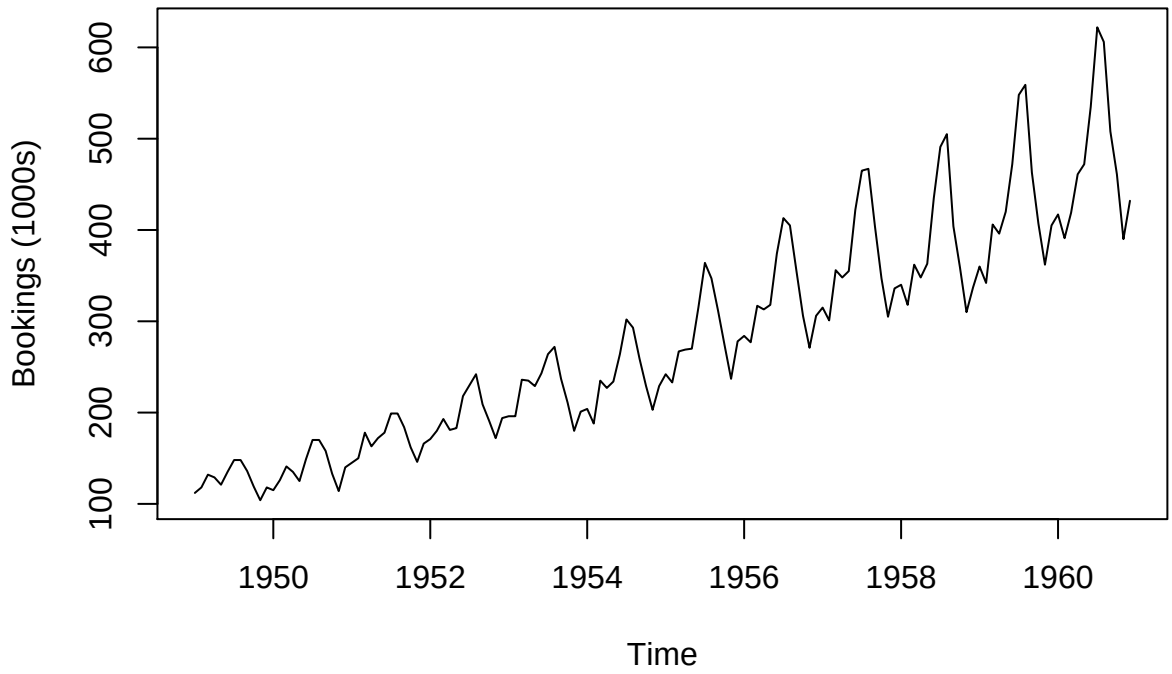
9.4 Comparing fitted SDE models 51

1 Concepts, terminology and examples

1.1 Plotting, manipulation, trends and seasonality

- A discrete time stochastic process is a collection of the same variable measured at different points in time. This is also know as a time series. We will always assume the data is observed at equidistant points in time (i.e. same time difference between consecutive observations).
- To get appropriate summaries and plots of the data we must tell R that the data is a time series, which is called a `ts` object in R. A built-in example is the dataset `AirPassengers` which we will abbreviate to `AP` to save typing:

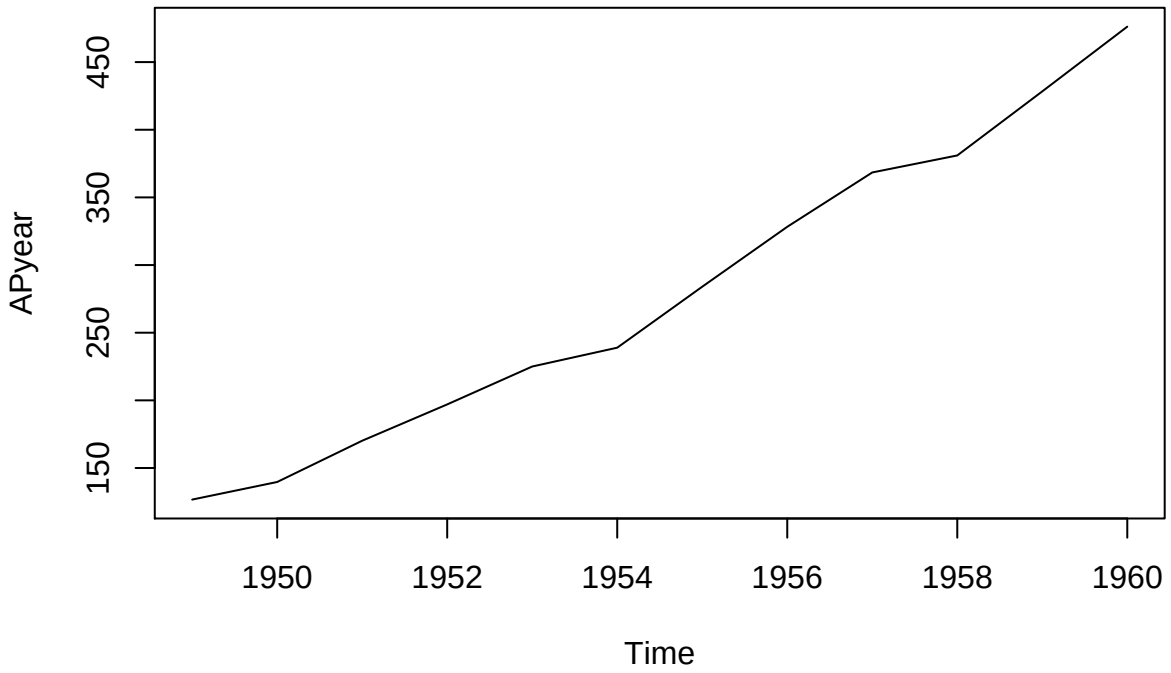
```
AP <- AirPassengers
plot(AP, ylab = "Bookings (1000s)")
```



- The generic function `plot` automatically detected that `AP` was a time series (`ts` object) and plotted the correct time on the horizontal axis (by calling `plot.ts` automatically).

- We can aggregate the time series over each year (sums the values by default, so we add `FUN = mean` to get monthly mean), which clearly shows an increasing trend in the yearly number of passenger bookings:

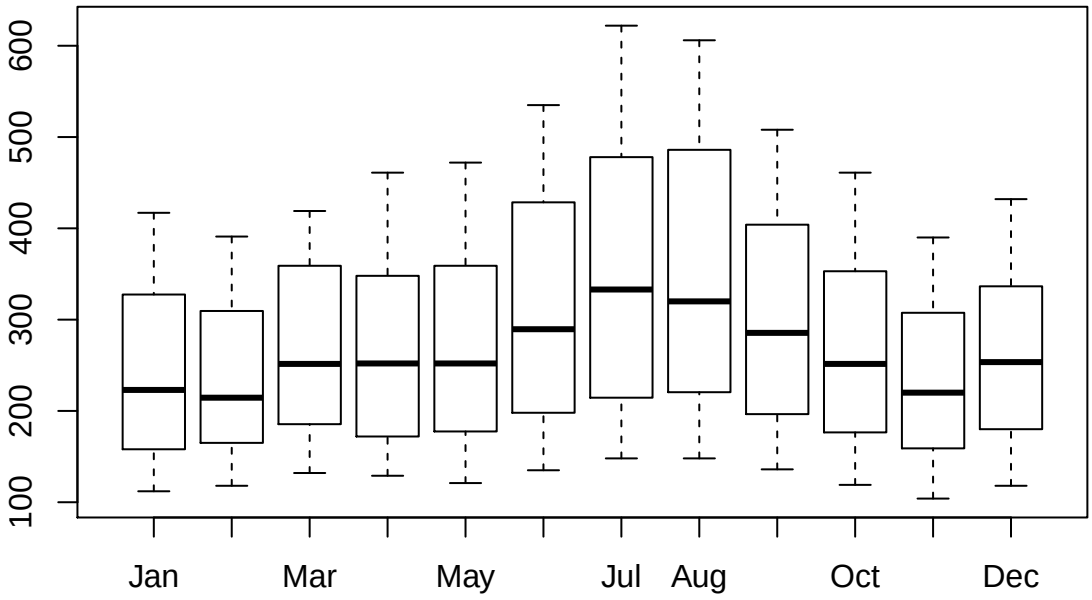
```
APyear <- aggregate(AP, FUN = mean)
plot(APyear)
```



Such a systematic change in a time series that is not periodic is known as the **trend** of the series.

- The repeating (periodic) pattern within each year seen in the original data is known as **seasonality** or **seasonal variation**. The function `cycle` indicates which cycle (aka season) each observation belongs to (in this case 1-12). A boxplot of the number of bookings for each cycle (month) clearly shows the seasonality:

```
cyc <- cycle(AP)
cyc <- factor(cyc, labels = month.abb)
boxplot(AP ~ cyc)
```



- If we only want to consider a subset of the data the function `window` is used:

```
APJun55Dec56 <- window(AP, start = c(1955, 6), end = c(1956, 12))
APJun55Dec56
```

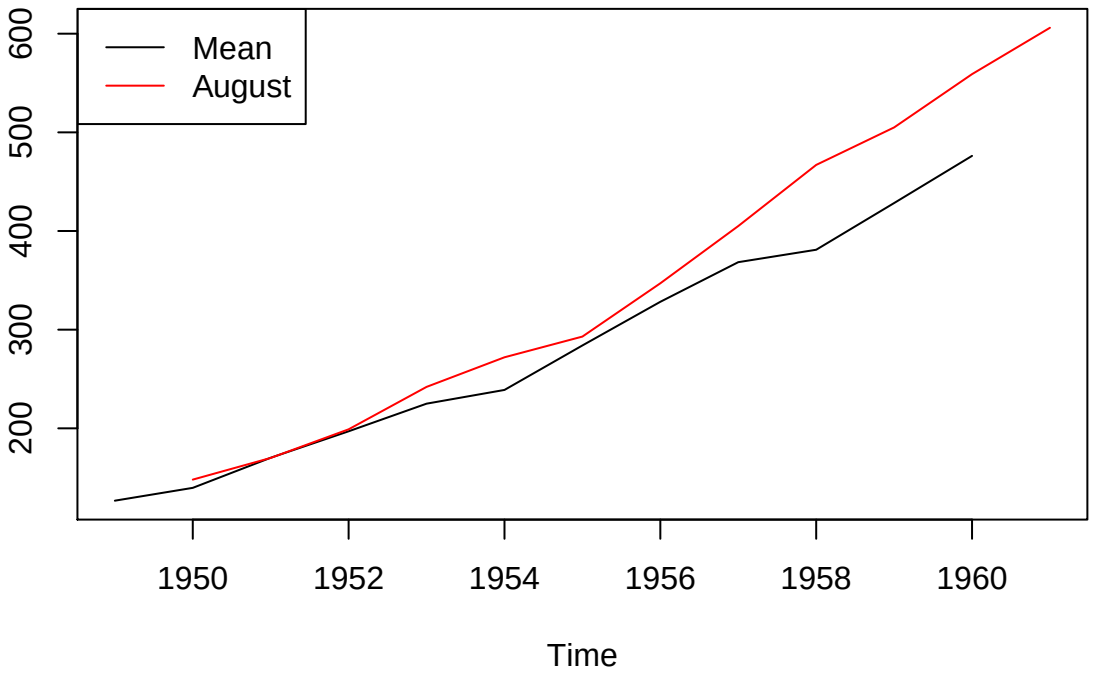
```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1955                     315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
```

- The window function can also sub-sample the time series according to the frequency:

```
APAug <- window(AP, start = c(1949, 8), end = end(AP), freq = TRUE)
```

- To plot multiple series in one plot we can use `ts.plot`:

```
ts.plot(APyear, APAug, col = c("black", "red"))
legend("topleft", legend = c("Mean", "August"), col = c("black", "red"), lty = 1)
```



1.2 Stochastic trend

- The trend for the passenger bookings data was very clear from the data (as well as the seasonal pattern), and a reasonable explanation may be increasing population, increasing prosperity and technological advances making tickets cheaper.
- In other situations the behavior of the trend may be less clear and we will need to handle it differently. For the quarterly exchange rate between GBP and NZD the picture is less clear:

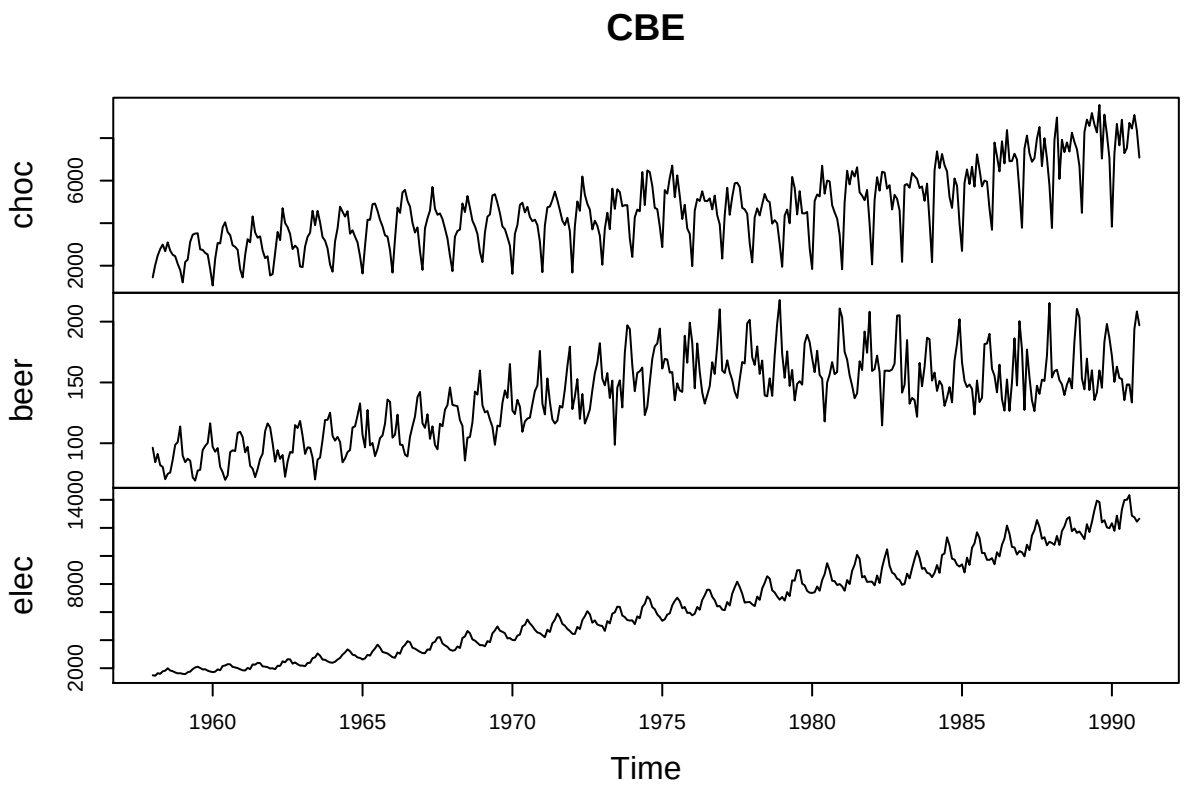
```r
www <- "https://asta.math.aau.dk/eng/static/datasets?file=pounds_nz.dat"
exchange_data <- read.table(www, header = TRUE)
exchange <- ts(exchange_data, start = 1991, freq = 4)
plot(exchange)
```

- In general it is dangerous to extrapolate the trend, and this is even more so the case for a series with a stochastic trend, which this example may have.
- Especially for financial time series stochastic trends with abrupt changes to the trend are common.

1.3 Multiple series

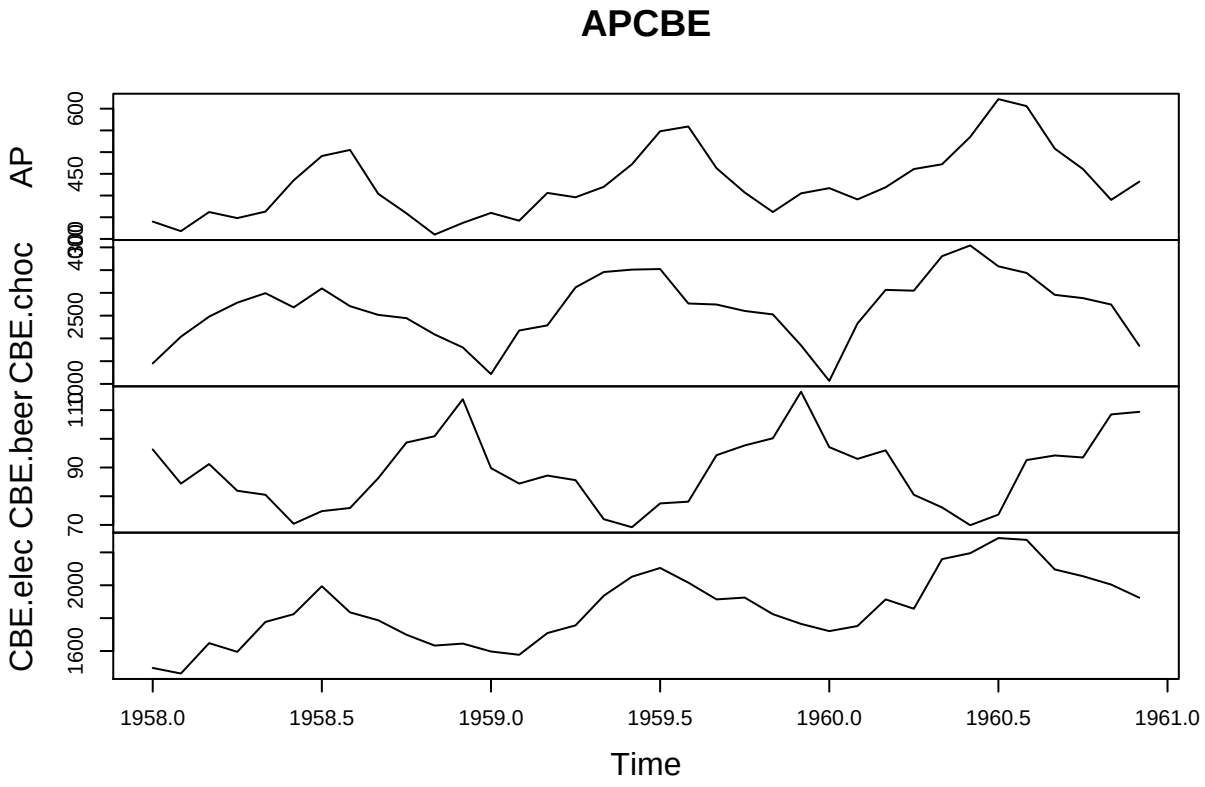
- Monthly time series from Jan. 1958 to Dec. 1990 of supply of three goods in Australia:
  - Electricity (Giga Watt hours)
  - Beer (Mega liters)
  - Chocolate (tonnes)

```r
CBEdata <- read.table("https://asta.math.aau.dk/eng/static/datasets?file=cbe.dat",
                      header = TRUE)
CBE <- ts(CBEdata, start = 1958, freq = 12)
plot(CBE)
```



- For overlapping time series **with the same frequency** we can get the data for the overlapping period like this:

```
APCBE <- ts.intersect(AP, CBE)
plot(APCBE)
```



- In the realm of time series people often find spurious correlations and mistake them for causal relationships. E.g. the number of passenger bookings in USA looks very similar to electricity consumption in Australia and there is a strong correlation:

```
cor(APCBE)
```

```
##                  AP   CBE.choc   CBE.beer   CBE.elec
## AP        1.0000000  0.6375808 -0.4434747  0.8841668
## CBE.choc  0.6375808  1.0000000 -0.6204467  0.7644756
## CBE.beer -0.4434747 -0.6204467  1.0000000 -0.3120179
## CBE.elec  0.8841668  0.7644756 -0.3120179  1.0000000
```

However, this does not imply that US air passengers really influence the electricity consumption in Australia (or vice versa)! Time series with similar trends and seasonality will typically show strong correlation even though they are unrelated. Better explanations may be due to similar population growth, seasonal patterns, etc.

1.4 Decomposition

1.4.1 Decomposition - trend term

- It can be useful to decompose the time series x_t into *the trend* m_t, *the seasonal variation* s_t, and *an error term* z_t. Here we will focus on an additive decomposition $x_t = m_t + s_t + z_t$, but multiplicative models $x_t = m_t \cdot s_t \cdot z_t$ are also available.

- To estimate the trend a simple solution is to use a moving average by averaging the preceding and following L values relative to the current point in time, where L is chosen to average out the season effect. So if the season effect is weekly we choose $L = 3$ such that

$$\hat{m}_t = (x_{t-3} + x_{t-2} + x_{t-1} + x_t + x_{t+1} + x_{t+2} + x_{t+3})/7$$

  E.g. if the current time t is a Tuesday the moving average averages today's value with the ones from the preceding Sat., Sun. and Mon. and the following Wed., Thu. and Fri. For an even season length like 12 months we have to do something else if we want an equal amount of past and future in the moving average, and we use half of the value six months in the past/future:

$$\hat{m}_t = \frac{\frac{1}{2}x_{t-6} + x_{t-5} + x_{t-4} + \cdots + x_0 + \cdots + x_{t+4} + x_{t+5} + \frac{1}{2}x_{t+6}}{12}$$

  E.g., the moving average for Jul. 1958 uses half the value in Jan. 1958 and half the value in Jan. 1959 and all the 11 values Feb.-Dec. 1958:

```
choc <- CBE[, "choc"]
i <- 7
(choc[i-6]/2 + sum(choc[(i-5):(i+5)]) + choc[i+6]/2)/12
```

```
## [1] 2413.625
```

This computation can be done for all time points (except the first six and last six) using `decompose`, which returns a list with a component `trend` (and other things to be discussed shortly):

```
choc_decomp <- decompose(choc)
choc_trend <- choc_decomp$trend
window(choc_trend, end = c(1958, 8))
```

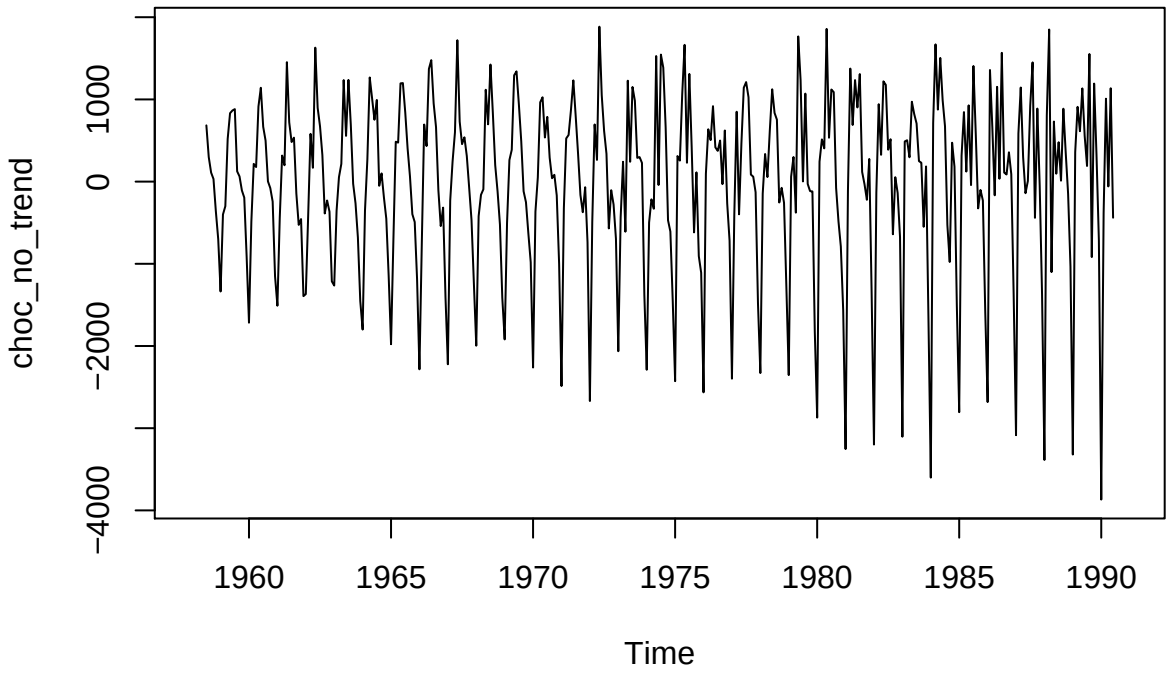
```
##           Jan      Feb      Mar      Apr      May      Jun      Jul
## 1958       NA       NA       NA       NA       NA       NA 2413.625
##           Aug
## 1958 2409.500
```

- Note that the procedure automatically chooses the appropriate size of the moving average based on the frequency of the time series, which was set to 12 in this case.

1.4.2 Decomposition - seasonal term

- To estimate the seasonal effect we subtract the trend and average for each period in the season (e.g. each month):

```
choc_no_trend <- choc - choc_trend
plot(choc_no_trend)
```




```
choc_jan <- mean(window(choc_no_trend, st = c(1958, 1), freq = TRUE), na.rm = TRUE)
choc_jan
```

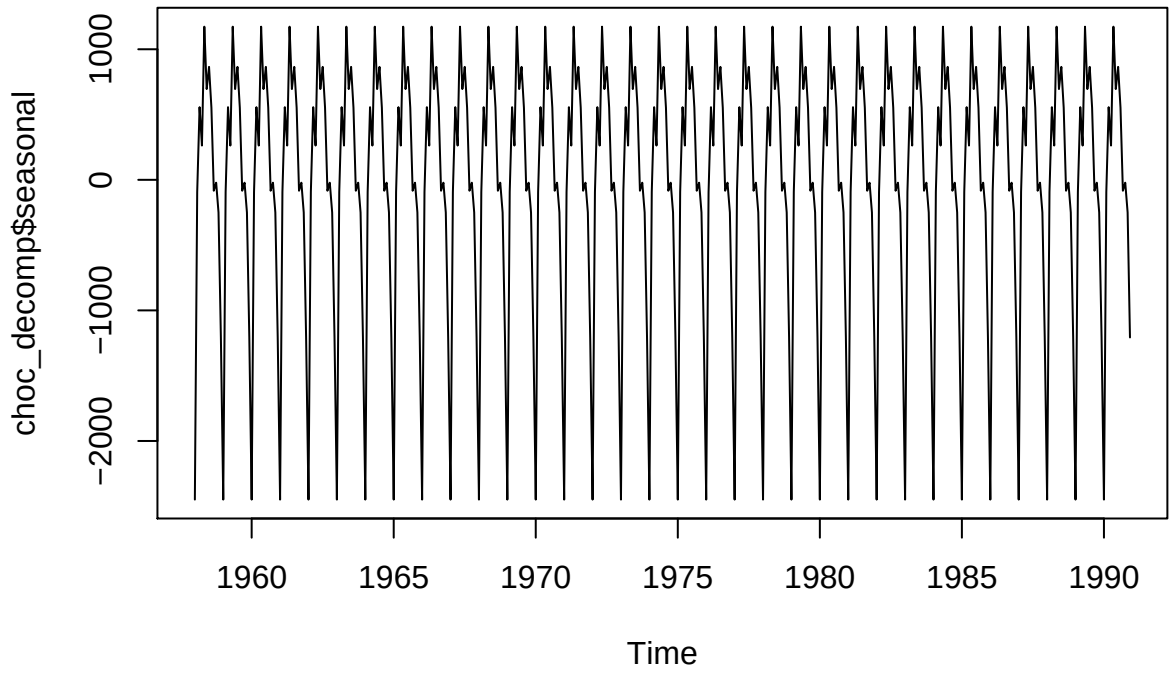
```
## [1] -2450.538
```

```
choc_may <- mean(window(choc_no_trend, st = c(1958, 5), freq = TRUE), na.rm = TRUE)
choc_may
```

```
## [1] 1171.215
```

It appears that chocolate consumption typically is much higher in May than in January. The season effects are also calculated by `decompose` and stored in the list as `seasonal`:

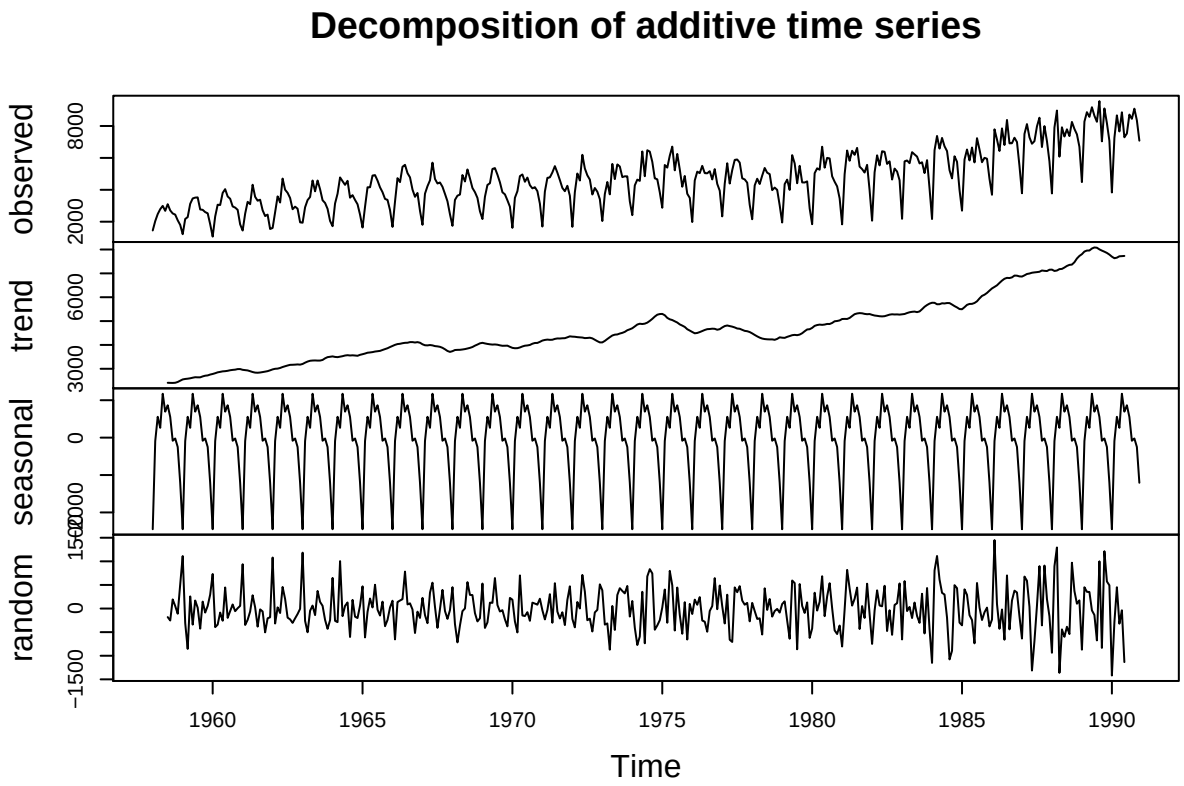
```
plot(choc_decomp$seasonal)
```



1.4.3 Decomposition - random term

- The entire decomposition can also be plotted:

```
plot(choc_decomp)
```

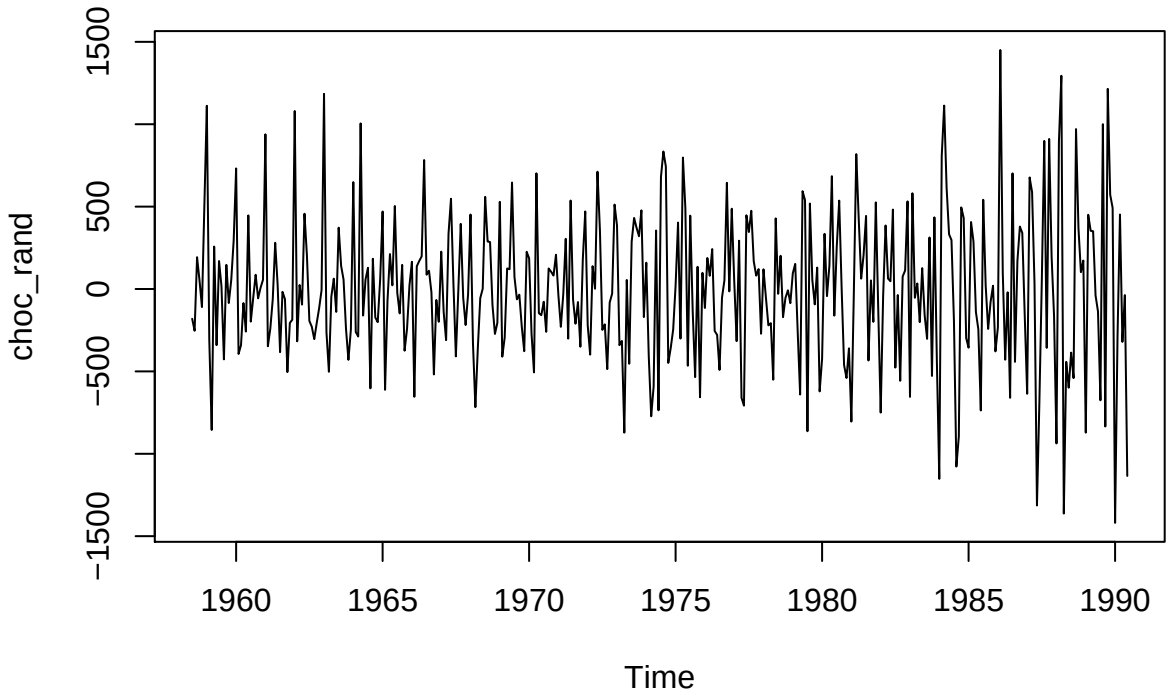


- Here the random term is simply given by $z_t = x_t - m_t - s_t$, and this can be accessed through the `random` entry of the decomposition output `choc_decomp$random`.

2 Stationarity and autocorrelation

- Consider the excess chocolate consumption data after adjusting for trend and season (and omitting the first six and last six entries which are `NA`):

```
choc_rand <- na.omit(choc_decomp$random)
plot(choc_rand)
```



This is a single realization of something we may think of as a random experiment. If things had been different (different weather, different chocolate advertisements, different economic fluctuations, ...) we could imagine another series.

This hypothetical scenario applies to any time series x_t where we imagine our dataset is somehow randomly seleceted among an entire ensemble of possible time series.

- Calculating the mean of all such hypothetical series would give us the mean function $\mu(t) = E(x_t)$ for all t. Since we already adjusted for trend and season we expect a mean value around zero at each time point, and we might assume $\mu(t) = \mu$ for all t, where μ is a fixed constant (zero in this case). If this is truly the case we say the series is stationary in the mean (first order stationary). In general we can estimate μ by the sample mean as we have been used to:

$$\hat{\mu} = \bar{x} = \frac{1}{n}\sum_{t=1}^{n} x_t.$$

- The variance (square of std. dev.) is in general also a function of time $\sigma^2(t) = E[(x_t - \mu(t))^2]$. If we assume we have stationarity of the variance also $\sigma^2(t) = \sigma^2$ we can estimate σ^2 by the sample variance

$$\hat{\sigma^2} = \frac{1}{n-1}\sum_{t=1}^{n}(x_t - \bar{x})^2.$$

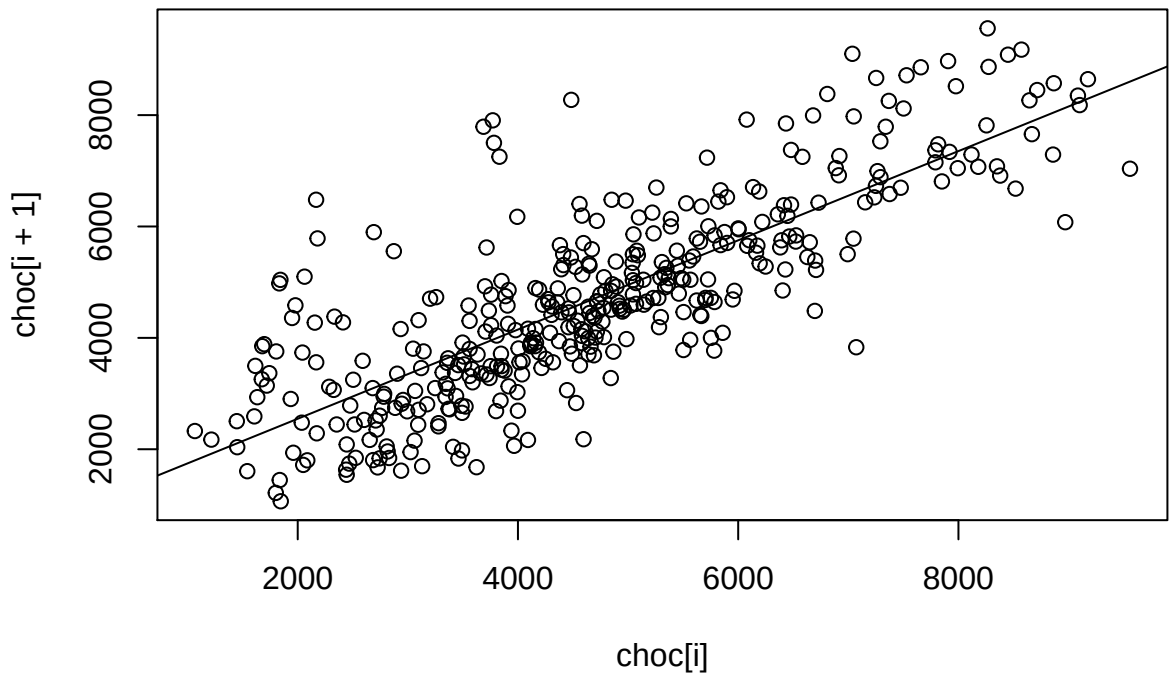
- Now consider a process that is stationary in the mean and variance. The variables at different times may be correlated and the process is called second order stationary if the covariance only depends on the number of time steps between the variables, $Cov(x_t, x_{t+k}) = \gamma(k)$ for all t, and γ is called the autocovariance function. The number of time steps k is called the lag. The autocovariance can be estimated from data:

$$\hat{\gamma}(k) = c_k = \frac{1}{n}\sum_{t=1}^{n-k}(x_t - \bar{x})(x_{t+k} - \bar{x})$$

- The more useful autocorrelation function (acf) is the normalized version that takes values between -1 and +1: $\rho(k) = \gamma(k)/\sigma^2$, which is estimated based on the estimate of the autocovariance above: $\hat{\rho}(k) = r_k = \frac{c_k}{c_0}$.

- The autocorrelation at lag 0 is always equal to 1, $\rho(0) = 1$ and $r_0 = 1$.
- For lag 1 it is the correlation between today's value x_t and tomorrows value x_{t+1}. If we look directly at the chocolate data this correlation is very strong (simply because there is an increasing trend, so high value is followed by a high value):

```
i <- 1:(length(choc)-1)
plot(choc[i], choc[i+1])
abline(lsfit(choc[i], choc[i+1]))
```

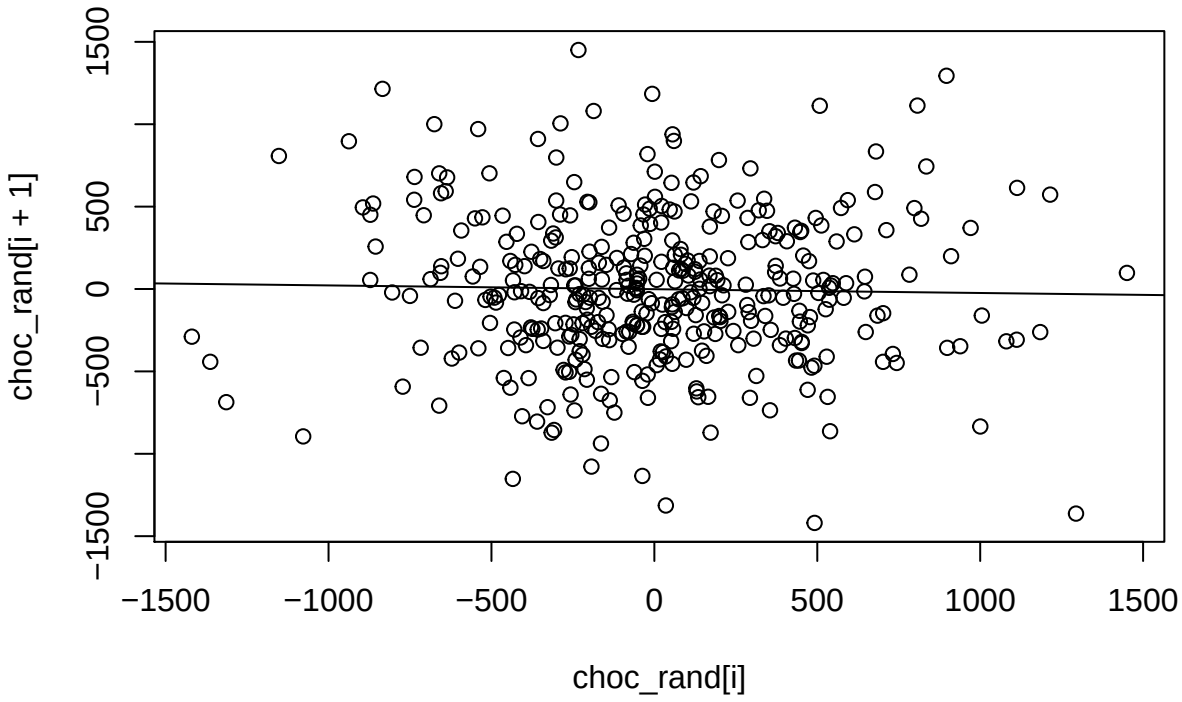



```
cor(choc[i], choc[i+1])
```

```
## [1] 0.8025919
```

- After removing the trend and seasonality there isn't really any dependence on the preceding value:

```
i <- 1:(length(choc_rand)-1)
plot(choc_rand[i], choc_rand[i+1])
abline(lsfit(choc_rand[i], choc_rand[i+1]))
```



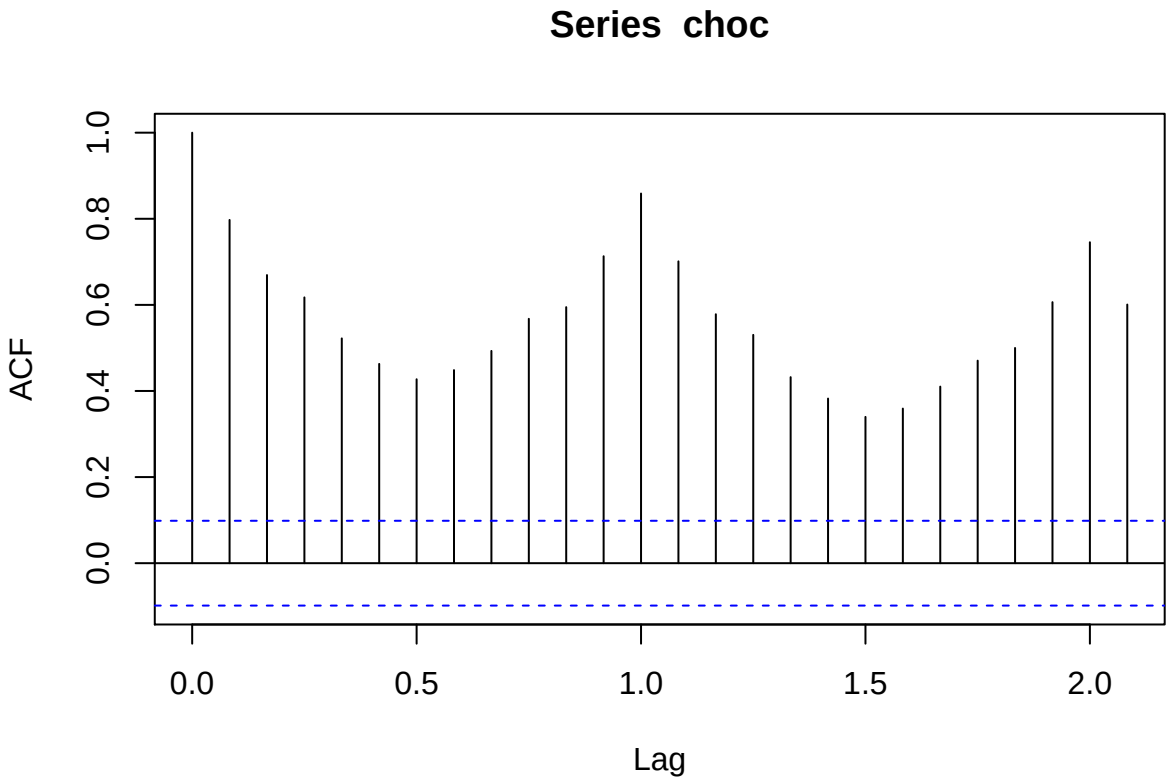

```
cor(choc_rand[i], choc_rand[i+1])
```

```
## [1] -0.02278652
```

2.1 Correlogram (empirical acf)

- To calculate the empirical autocorrelation at many lags we use `acf` which produces a plot by default:

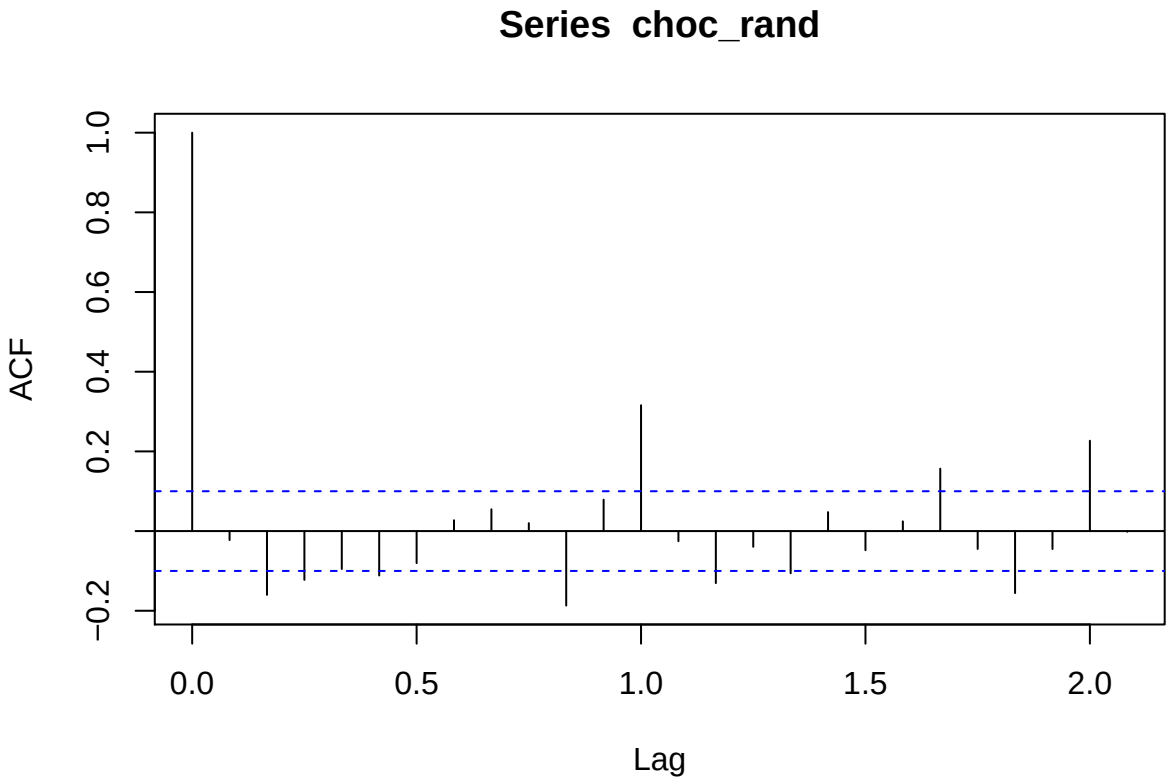
```
choc_acf <- acf(choc)
```



This is called the correlogram.

- It is possible to calculate the correlogram for any time series (also non-stationary). However, the theoretical properties only make sense for a stationary process.
- The correlogram value at lag k, r_k is an estimate of the true autocorrelation $\rho(k)$.
- In particular if $\rho(k) = 0$ then r_k will usually not be exactly zero. Rather it is approximately normally distributed with a mean close to zero and standard deviation $1/\sqrt{n}$, where n is the length of the series. Based on this approximate significance bands are drawn at $\pm 1.96/\sqrt{n}$.
- The significance bands are only valid for a single lag! Even if $\rho(k) = 0$ for all lags $k = 1, 2, \ldots$ we expect $1/20$ of the r_k's to fall outside the line. Furthermore the estimates are heavily correlated so if one falls outside it is also more likely that the one next does.
- It is typical to look at a few specific lags. E.g. lag 1 for the immediate past and lag 12 if a season of length 12 is expected. A spike at lag 12 indicates that there is a seasonal pattern that has not been adequately accounted for. It appears the seasonal adjustment for the chocolate data hasn't been perfect since there is a somewhat large autocorrelation at lag 12 (1 year):

```
acf(choc_rand)
```

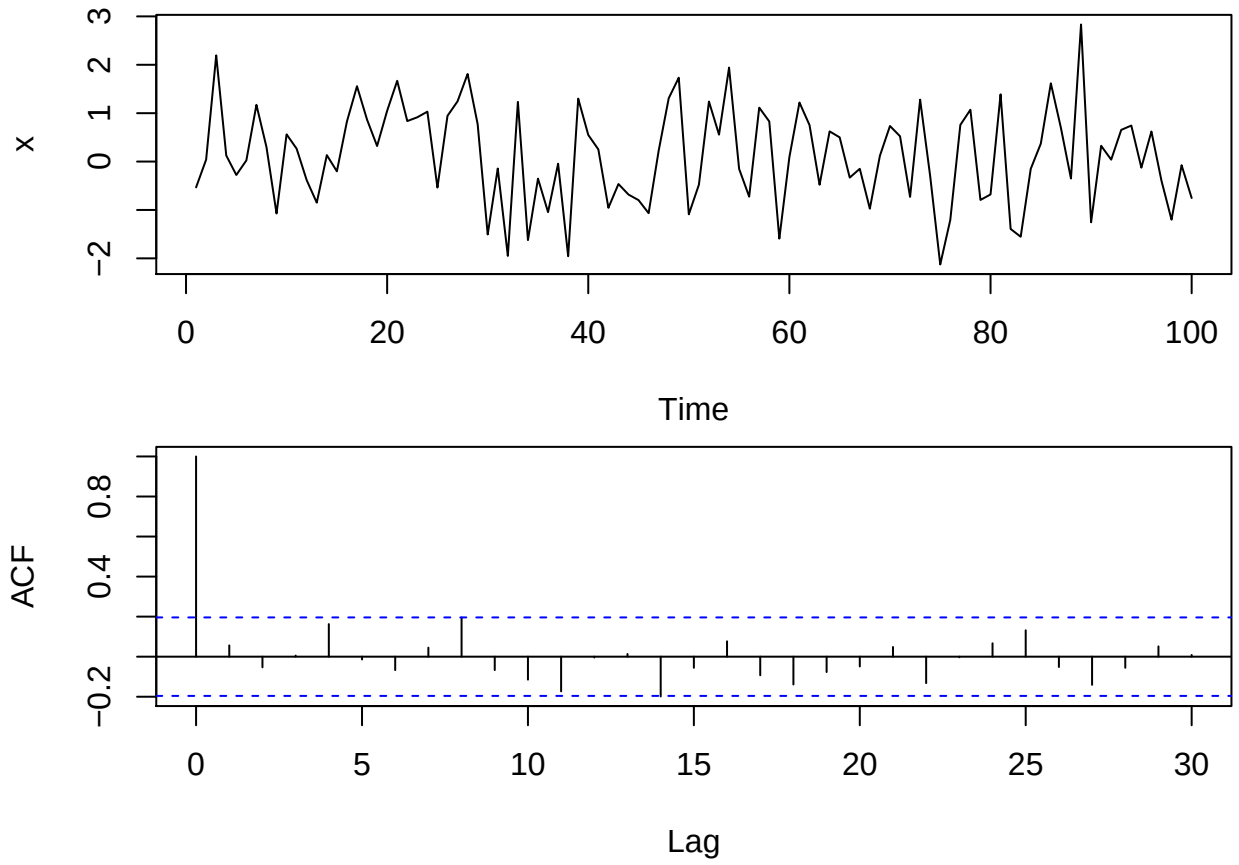


2.2 Typical patterns in correlograms

It can be illustrative to consider a few specific examples of correlograms:

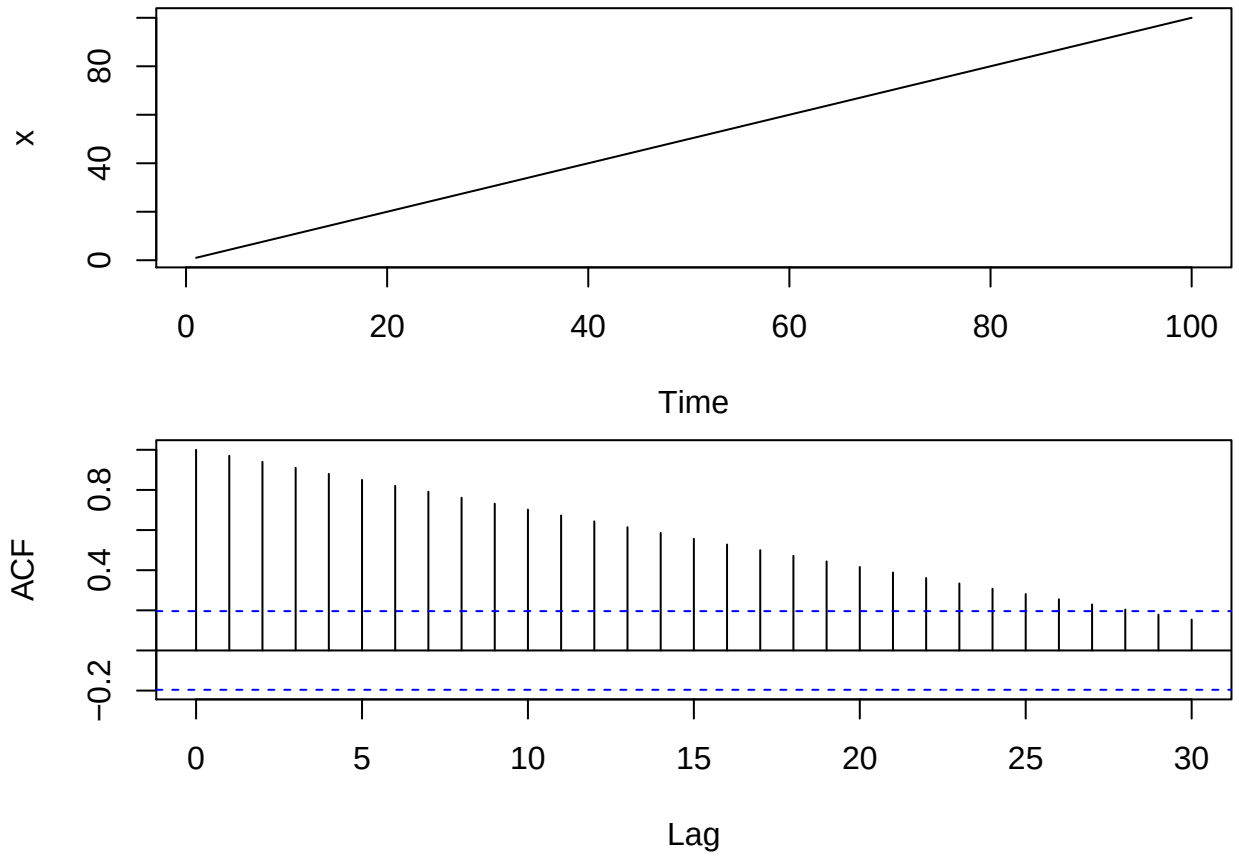
- Completely random:

```
x <- rnorm(100)
par(mfrow = c(2,1), mar = c(4,4,0.5,0.5))
ts.plot(x)
acf(x, main = "", lag.max = 30)
```



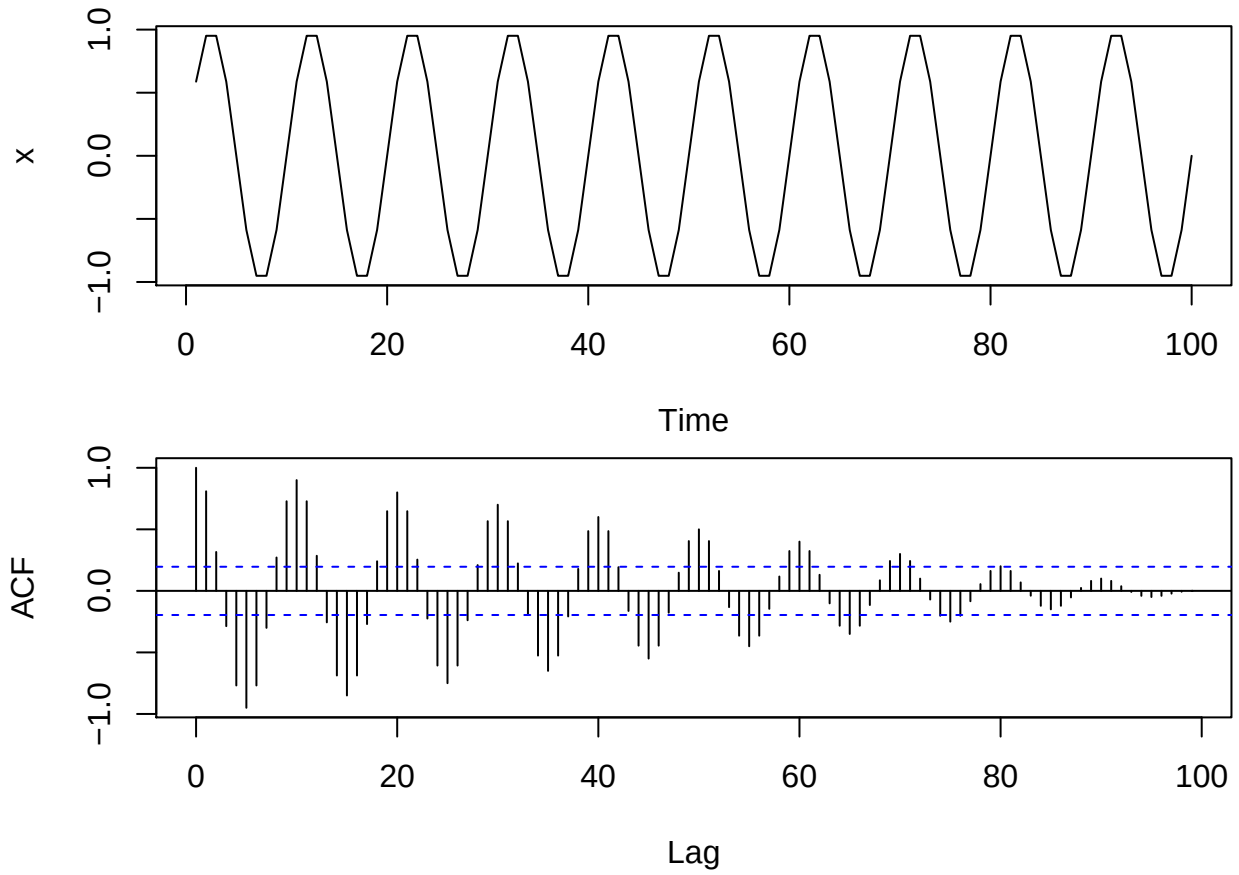
- Trend only:

```
x <- 1:100
par(mfrow = c(2,1), mar = c(4,4,0.5,0.5))
ts.plot(x)
acf(x, main = "", lag.max = 30)
```



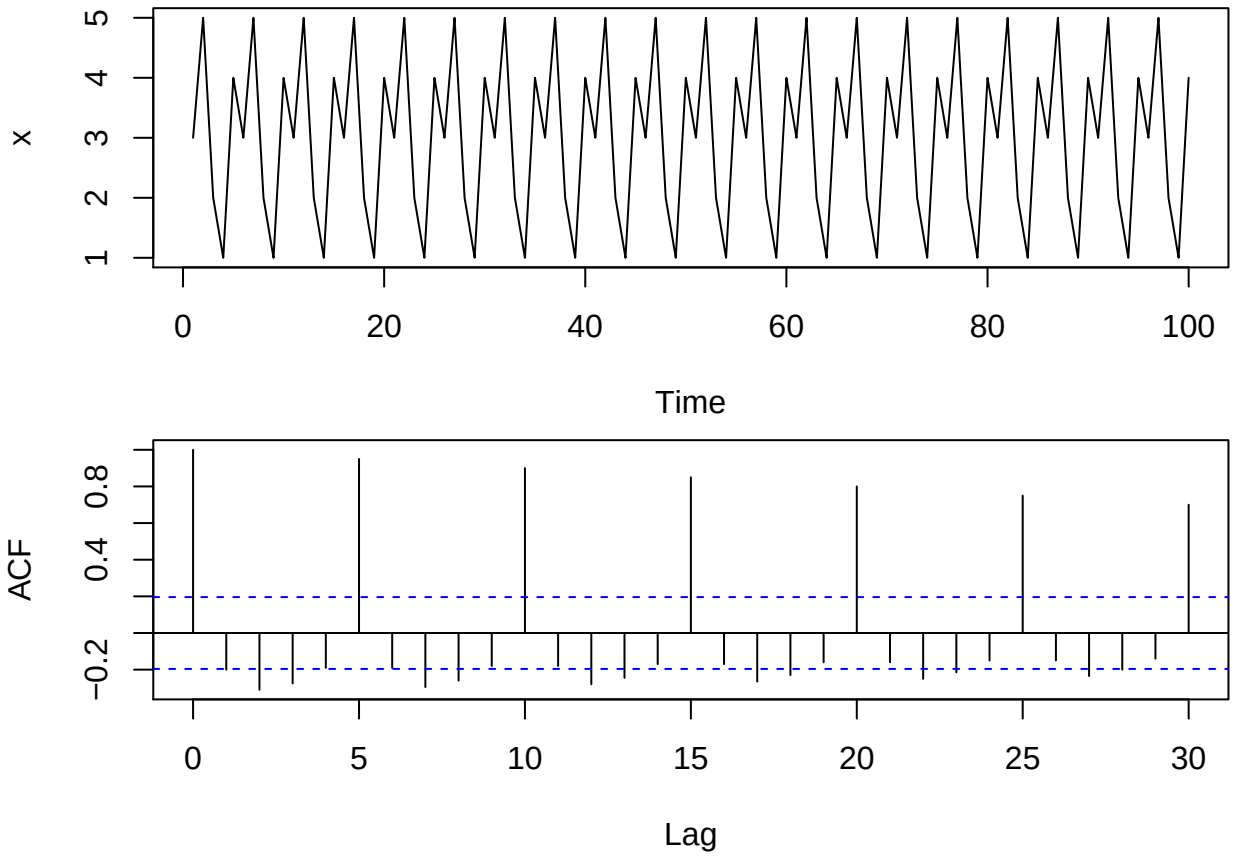
- Sinusoidal wave:

```
x <- sin((1:100)*2*pi/10)
par(mfrow = c(2,1), mar = c(4,4,0.5,0.5))
ts.plot(x)
acf(x, main = "", lag.max = 100)
```



- Repeated pattern:

```
x0 <- c(3,5,2,1,4)
x <- rep(x0, 20)
par(mfrow = c(2,1), mar = c(4,4,0.5,0.5))
ts.plot(x)
acf(x, main = "", lag.max = 30)
```



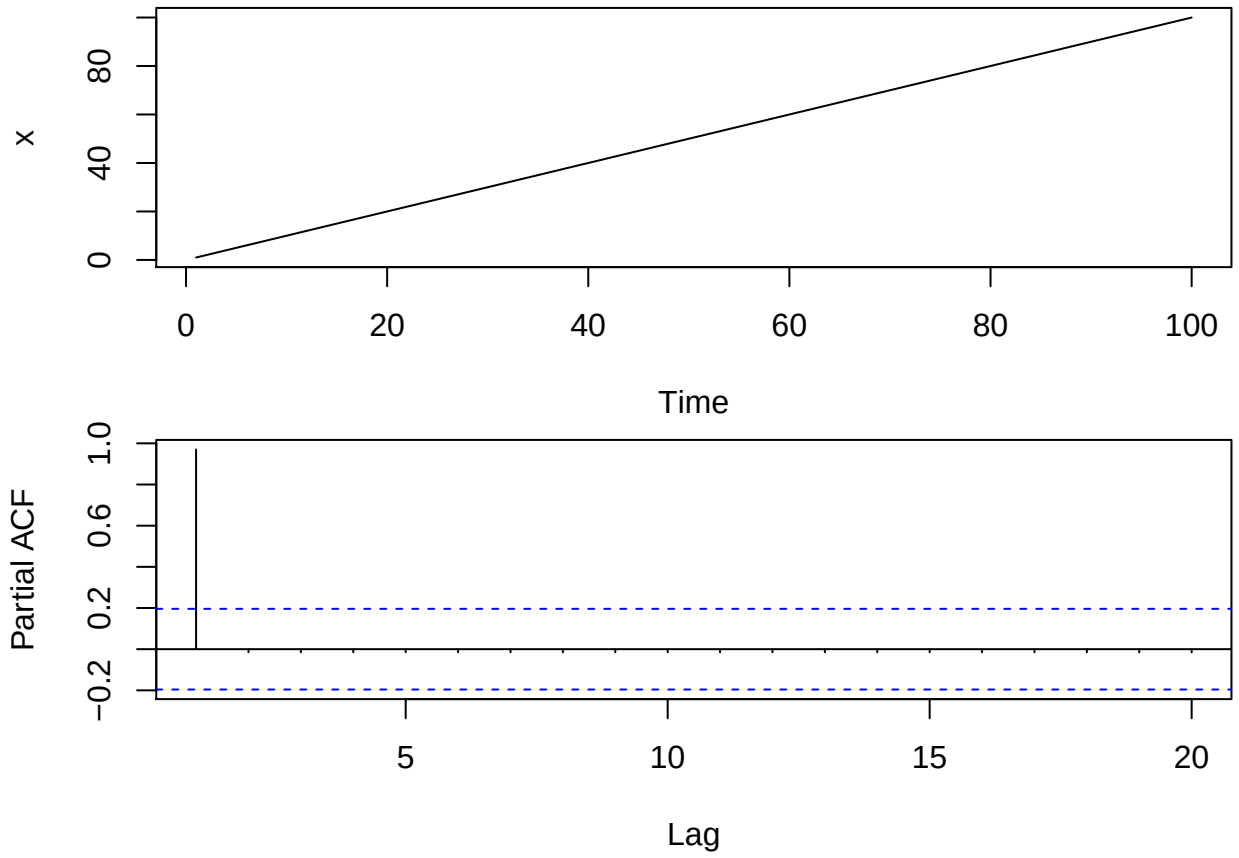
2.3 Partial autocorrelation function

- If a time series has a strong correlation with the values lagged by one time step it most likely also has a strong correlation with the values lagged by two time steps: When today's value is highly influenced by yesterdays, then yesterdays will be highly influenced by the day before, and thus the lag 2 value influences the current value through the lag 1 value.
- The partial autocorrelation (pacf) at lag 2 is the correlation between the time series and the lag 2 values **after controlling for the lag 1 values**.
- This is analogous to the multiple regression setup

$$x_t = \beta_0 + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \epsilon$$

  where β_2 is a partial effect of the corresponding predictor (x_{t-2}) **when controlling for the other predictor** (x_{t-1}).
- In general the pacf at lag k is the correlation of x_t and x_{t-k} after controlling for the intermediate variables $x_{t-1}, \ldots, x_{t-k-1}$.
- A more detailed description of partial correlation in the multiple regression setup is in Section 11.7 of Agresti (2013).
- For a strong deterministic trend the pacf only shows strong correlation in the first lag and after controlling for this the following lags show no extra correlation:

```
x <- 1:100
par(mfrow = c(2,1), mar = c(4,4,0.5,0.5))
ts.plot(x)
pacf(x, main = "")
```



3 Basic stochastic models

3.1 White noise

Suppose y_t is a time series and we have a model which gives us predictions $\hat{y}_t$. Then the residuals are $x_t = y_t - \hat{y}_t$ for each time point $t = 1, \ldots, n$. If the model has correctly captured the trend, seasonality and any autocorrelation in the original process y_t then the correlogram (acf) of the residual process x_t should show no obvious patterns. Such a residual process with no autocorrelation or other structure is called *white noise*.

3.1.1 Definition and properties of *white noise*

A time series w_t, $t = 1, \ldots, n$ is *white noise* if the variables $w_1, w_2, \ldots, w_n$ are *independent* and *identically* distributed with a mean of zero.

From the definition it follows that white noise is a second order stationary process since the variance function $\sigma^2(t) = \sigma^2$ is the same constant for all t and the autocovariance is $Cov(w_t, w_{t+k}) = 0$ for all $k \neq 0$ which does not depend on t. We summaries this as:

$$\mu = 0$$

$$\gamma(k) = Cov(w_t, w_{t+k}) = \begin{cases} \sigma^2 \text{ for } k = 0 \\ 0 \text{ for } k \neq 0 \end{cases}$$

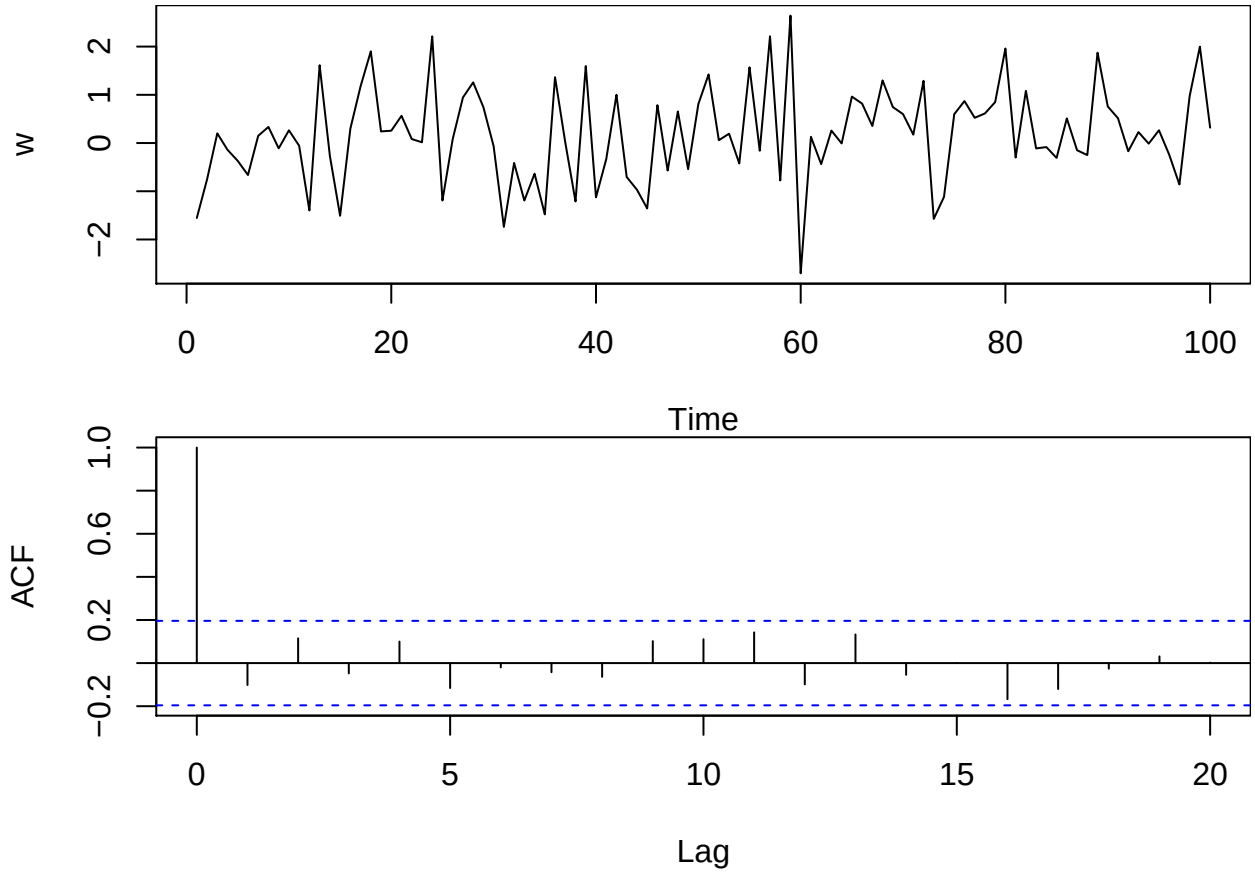
$$\rho(k) = \begin{cases} 1 \text{ for } k = 0 \\ 0 \text{ for } k \neq 0 \end{cases}$$

Often we will also assume the distribution of each w_t is Gaussian (i.e. $w_t \sim N(0, \sigma)$) and then we call it *Gaussian white noise*.

3.1.2 Simulation of white noise

To understand how white noise behaves we can simulate it with R and plot both the series and the autocorrelation:

```
w <- rnorm(100, mean = 0, sd = 1)
par(mfrow = c(2,1), mar = c(4,4,0,0))
ts.plot(w)
acf(w)
```



It is a good idea to repeat this simulation and plot a few times to appreciate the variablility of the results.

3.2 Random walk

A time series x_t is called a random walk if

$$x_t = x_{t-1} + w_t$$

where w_t is a white noise series. Using $x_{t-1} = x_{t-2} + w_{t-1}$ we get

$$x_t = x_{t-2} + w_{t-1} + w_t$$

Substituting for x_{t-2} we get

$$x_t = x_{t-3} + w_{t-2} + w_{t-1} + w_t$$

Continuing this way we would get an infinite sum of white noise

$$x_t = w_t + w_{t-1} + w_{t-2} + w_{t-3} + \dots$$

However, we will assume we have a fixed starting point $x_0 = 0$ such that

$$x_t = w_1 + w_2 + \cdots + w_t$$

3.2.1 Properties of random walk

A random walk x_t has a constant mean function

$$\mu(t) = 0$$

since the random walk at time t is a sum of t white noise terms that all have mean zero.

However, the variance function

$$\sigma^2(t) = t \cdot \sigma^2$$

clearly depends on the time t, so the process is not stationary. The variance function is derived from the general fact that for **independent random variables**, y_1 and y_2, the variance of the sum is

$$Var(y_1 + y_2) = Var(y_1) + Var(y_2).$$

Thus,

$$Var(x_t) = Var(w_1 + w_2 + \cdots + w_t) = \sigma^2 + \sigma^2 + \cdots + \sigma^2 = t\sigma^2$$

The non-stationary autocovariance function is

$$Cov(x_t, x_{t+k}) = t\sigma^2, \quad k = 0, 1, \dots$$

which only depends on how many white noise terms x_t and x_{t+k} have in common (t) and not how far they are separated (k).

By combining the two results we obtain the non-stationary autocorrelation function

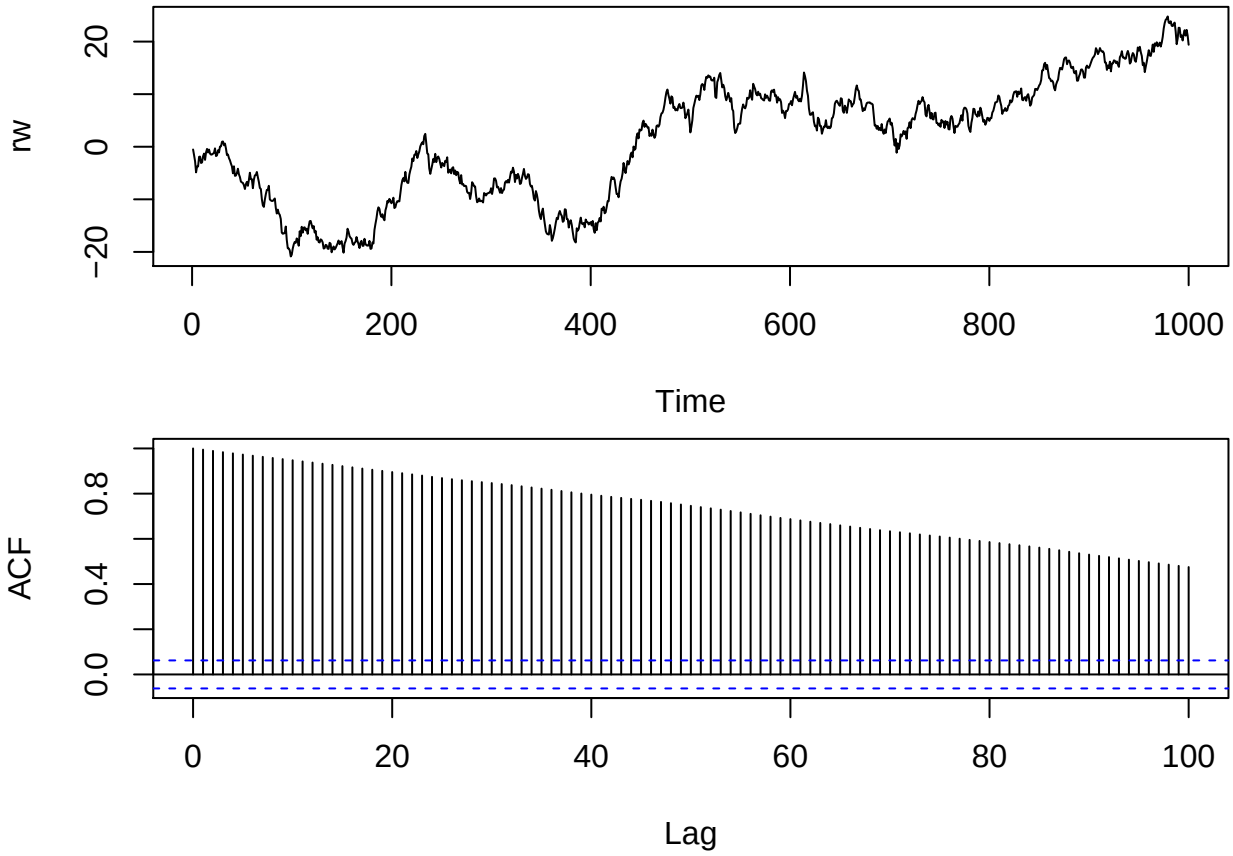
$$Cor(x_t, x_{t+k}) = \frac{Cov(x_t, x_{t+k})}{\sqrt{Var(x_t)Var(x_{t+k})}} = \frac{t\sigma^2}{\sqrt{t\sigma^2(t+k)\sigma^2}} = \frac{1}{\sqrt{1+k/t}}$$

When t is large compared to k we have very high correlation (close to one) and even though the process is not stationary we expect the correlogram of a reasonably long random walk to show very slow decay.

3.2.2 Simulation of random walk

We already know how to simulate Gaussian white noise (with `rnorm`) and the random walk is just a cumulative sum of white noise:

```
w <- rnorm(1000)
rw <- cumsum(w)
par(mfrow = c(2,1), mar = c(4,4,0.5,0.5))
ts.plot(rw)
acf(rw, lag.max = 100)
```



3.2.3 Differencing

The slowly decaying acf for random walk is a classical sign of non-stationarity, indicating there may be some kind of trend. In this case there is no real trend, since the theoretical mean is constant zero, but we refer to the apparent trend which seems to change directions unpredictiably as a stochastic trend.

If a time series shows these signs of non-stationarity we can try to study the time series of differences and see if that is stationary and easier to understand:

$$\nabla x_t = x_t - x_{t-1}.$$

Since we assume/define $x_0 = 0$ we get

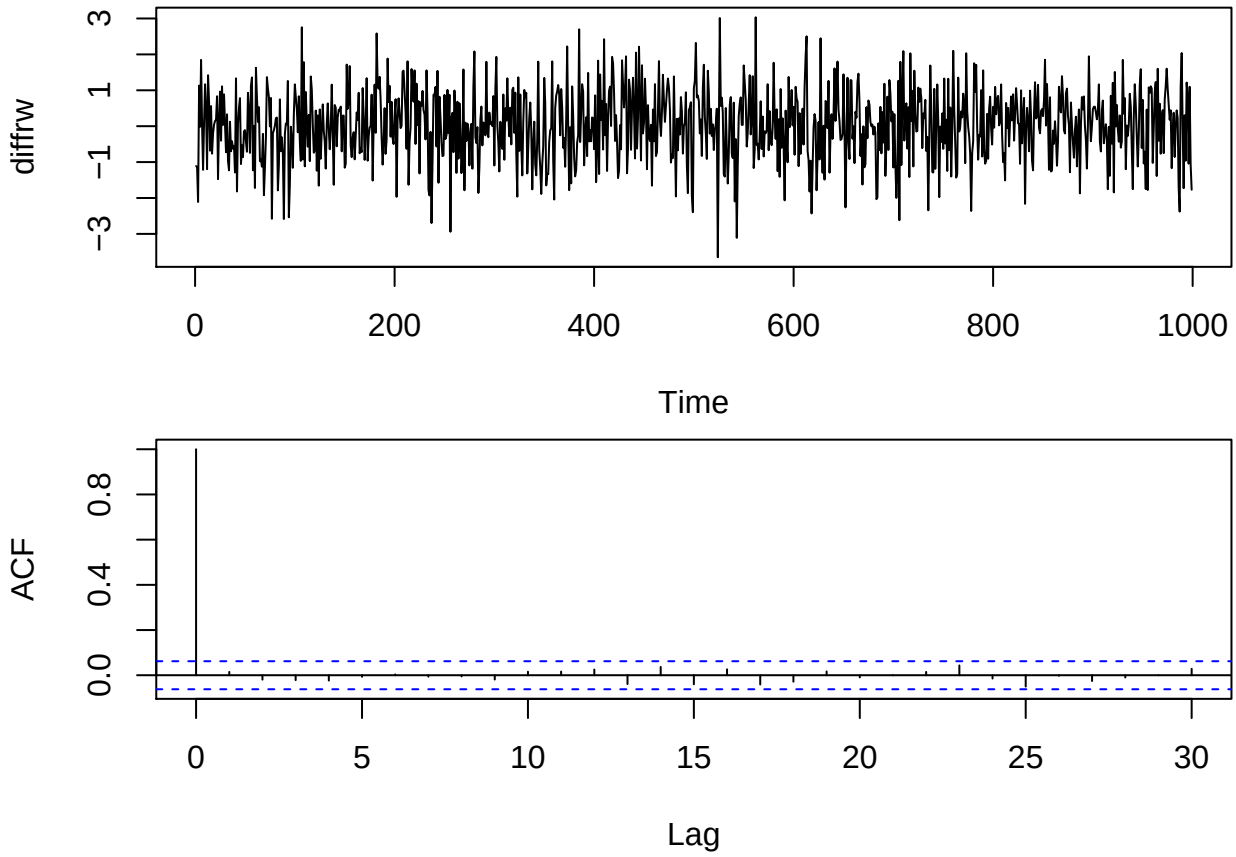
$$\nabla x_1 = x_1$$

$$\nabla x_2 = x_2 - x_1$$
$$\nabla x_3 = x_3 - x_2$$

etc.

Specifically when we difference a random walk we recover the white noise series $\nabla x_t = w_t$:

```
diffrw <- diff(rw)
par(mfrow = c(2,1), mar = c(4,4,0.5,0.5))
ts.plot(diffrw)
acf(diffrw, lag.max = 30)
```

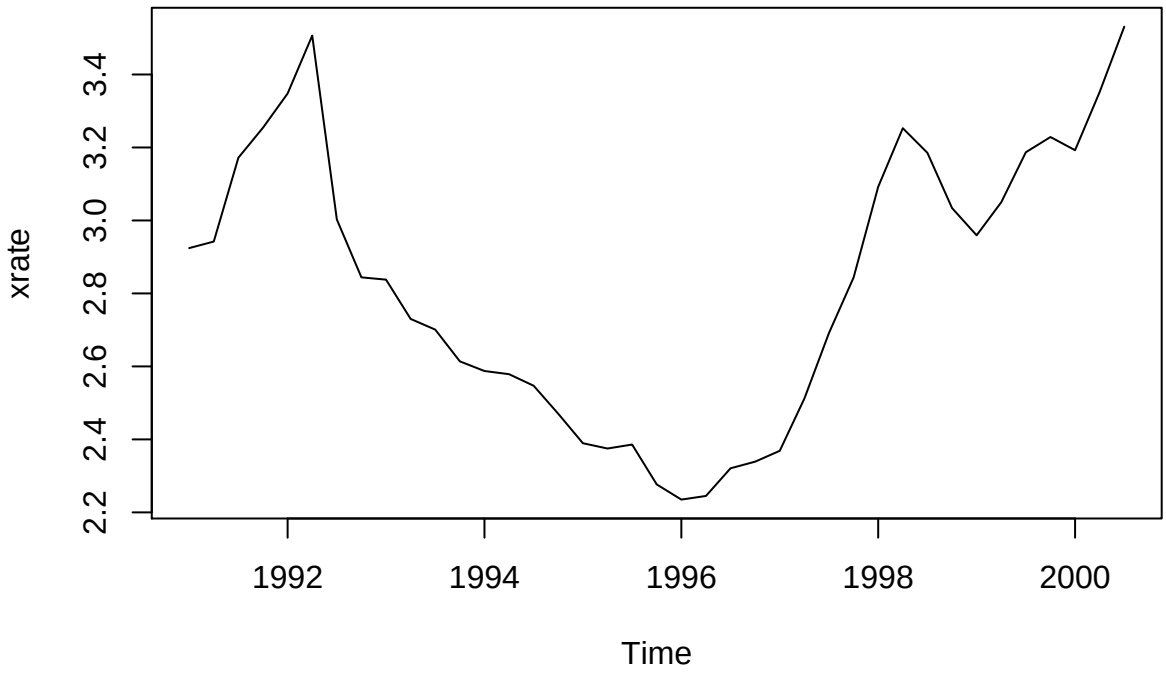


In general if a time series needs to be differenced to become stationary we say that the series is integrated (of order 1).

3.2.4 Example: Exchange rate

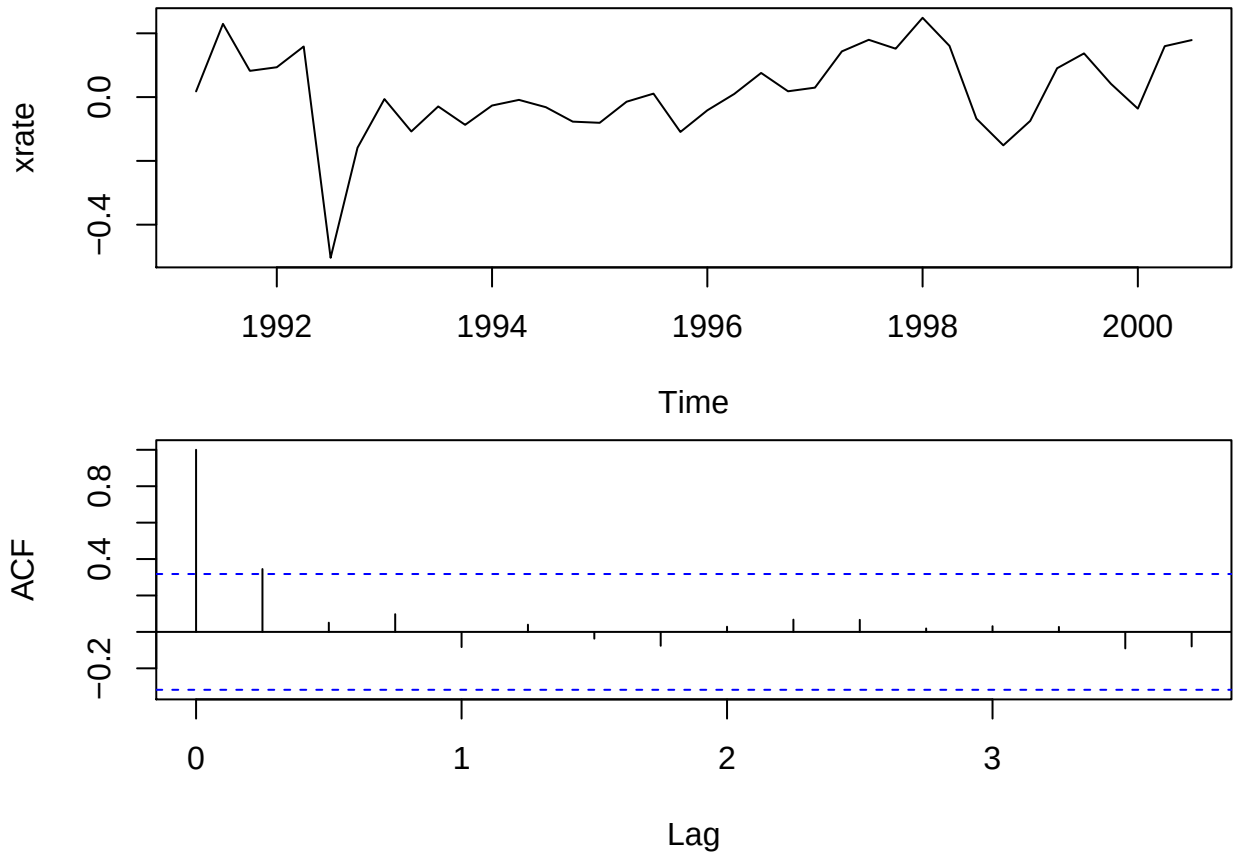
We have previously studied the exchange rate from GBP to NZD and observed what looked like an unpredictable stochastic trend and we would like to see if it could reasonably be described as a random walk.

```
www <- "https://asta.math.aau.dk/eng/static/datasets?file=pounds_nz.dat"
exchange_data <- read.table(www, header = TRUE)
exchange <- ts(exchange_data, start = 1991, freq = 4)
plot(exchange)
```



To this end we difference the series and see if the difference looks like white noise:

```
diffexchange <- diff(exchange)
par(mfrow = c(2,1), mar = c(4,4,0.5,0.5))
plot(diffexchange)
acf(diffexchange)
```



The first order difference looks reasonably stationary, so the original exchange rate series could be considered integrated of order 1. However, there is an indication of significant autocorrelation at lag 1, so a random walk might not be a completely satisfactory model for this dataset.

4 Auto-regressive (AR) models

4.1 Auto-regressive model of order 1: AR(1)

A significant auto-correlation at lag 1 means that x_t and x_{t-1} are correlated so the previous value x_{t-1} can be used to predict the current value x_t. This is the idea behind an auto-regressive model of order one AR(1):

$$x_t = \alpha_1 x_{t-1} + w_t$$

where w_t is white noise and the auto-regressive coefficient α_1 is a parameter to be estimated from data.

The model is only stationary if $-1 < \alpha_1 < 1$ such that the dependence of the past decreases with time.

4.1.1 Properties of AR(1) models

For a stationary AR(1) model with $-1 < \alpha_1 < 1$ it can be shown that

- $\mu(t) = 0$
- $Var(x_t) = \sigma^2(t) = \sigma^2/(1 - \alpha_1^2)$

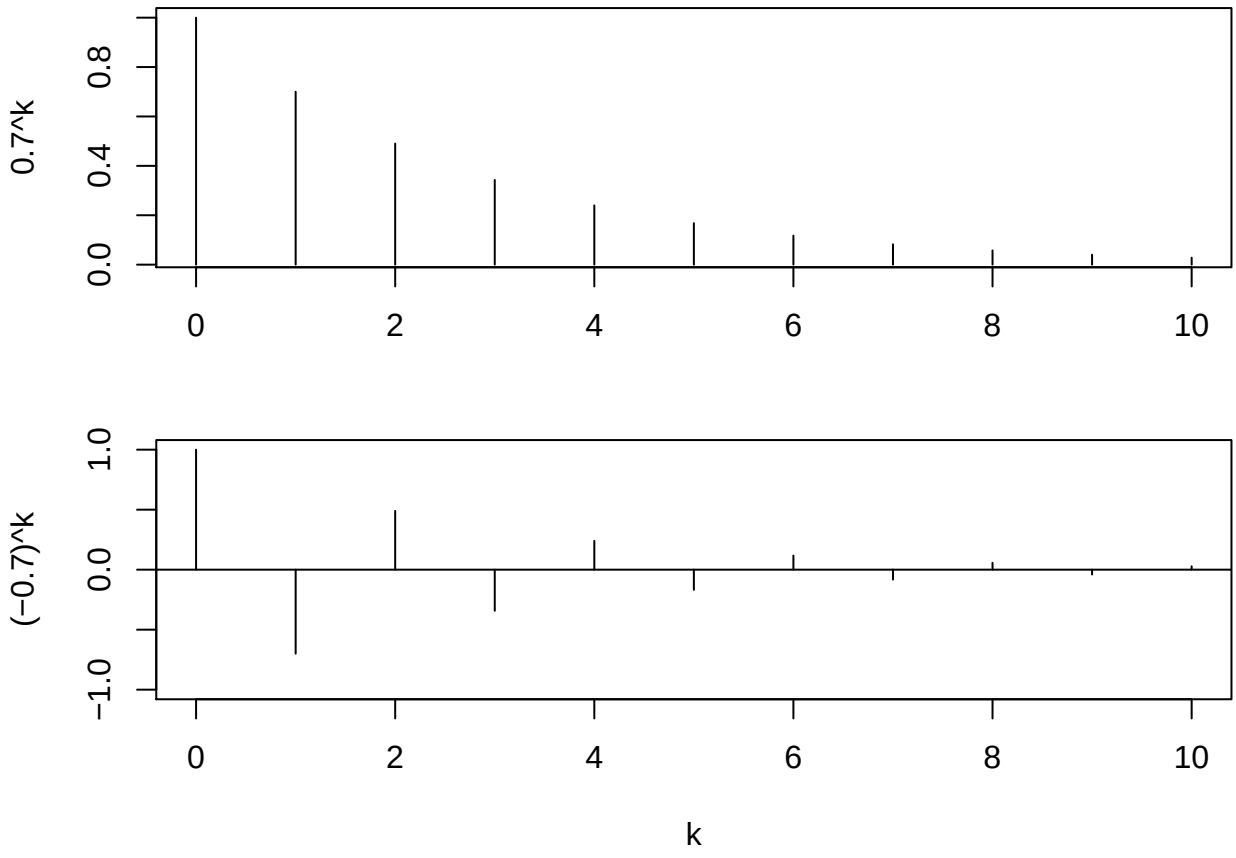
- $\gamma(k) = \alpha_1^k \sigma^2/(1-\alpha_1^2)$
- $\rho(k) = \alpha_1^k$

Furthermore, the partial autocorrelation of an AR(1) model is zero for all lags $k \geq 2$.

Below are the theoretical correlograms for the following AR(1) models:

- Model 1: $x_t = 0.7x_{t-1} + w_t$
- Model 2: $x_t = -0.7x_{t-1} + w_t$

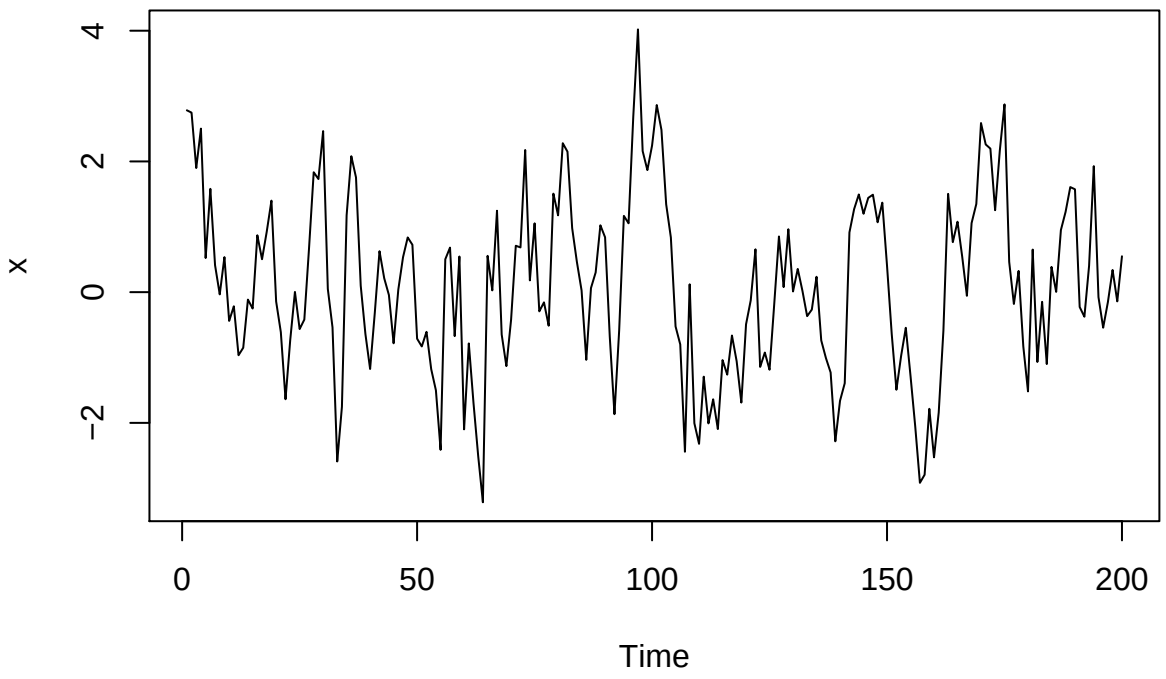
```
k <- 0:10
par(mfrow = c(2,1), mar = c(4,4,0.5,0.5))
plot(k, 0.7^k, type = "h", xlab = "")
plot(k, (-0.7)^k, type = "h", ylim = c(-1,1))
abline(h=0)
```



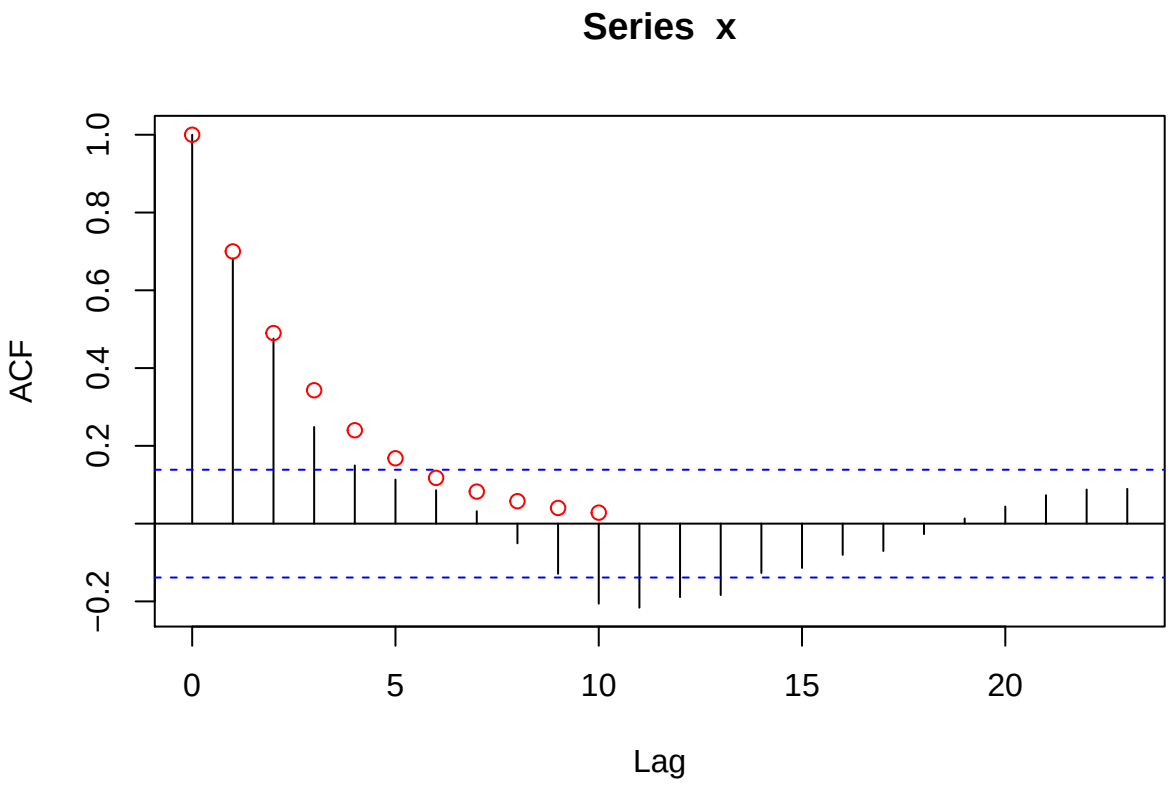
4.1.2 Simulation of AR(1) models

R has a built-in function `arima.sim` to simulate AR(1) and other more complicated models called ARMA and ARIMA. It needs the model (i.e. the autoregressive coefficient α_1) and the desired number of time steps `n`. To simulate 200 time steps of AR(1) with $\alpha_1 = 0.7$ we do:

```r
x <- arima.sim(model = list(ar = 0.7), n = 200)
plot(x)
```




```r
acf(x)
points(k, 0.7^k, col = "red")
```



Here we have compared the empirical correlogram with the theoretical values of the model.

4.1.3 Fitted AR(1) models

To estimate the autoregressive parameter α_1 we use the function `ar`:

```
fit <- ar(x, order.max = 1)
```

The resulting object contains the value of the estimated parameter and a bunch of other information. In this case the input data are artificial so we know we should ideally get a value close to 0.7:

```
fit$ar
```

```
## [1] 0.6814584
```

An estimate of the variance of the estimate $\hat{\alpha}_1$ is given in `fit$asy.var.coef` (the estimated std. error is the square root of this):

```
fit$asy.var.coef
```

```
##              [,1]
## [1,] 0.002705124
```

```
se <- sqrt(fit$asy.var.coef)
se
```

```
##            [,1]
## [1,] 0.0520108
```

```
ci <- c(fit$ar - 2*se, fit$ar + 2*se)
ci
```

```
## [1] 0.5774368 0.7854800
```

There are several different methods that can be used to estimate the model, but we ignore the details of this. However, first step in the estimation is to subtract the mean $\bar{x}$ of the time series before doing anything else, so the model that is fitted is actually:

$$x_t - \bar{x} = \alpha_1 \cdot (x_{t-1} - \bar{x}) + w_t$$

To predict the value of x_t given x_{t-1} we use that w_t is white noise so we expect it to be zero on average:

$$\hat{x}_t - \bar{x} = \hat{\alpha}_1 \cdot (x_{t-1} - \bar{x})$$

So the predictions are given by

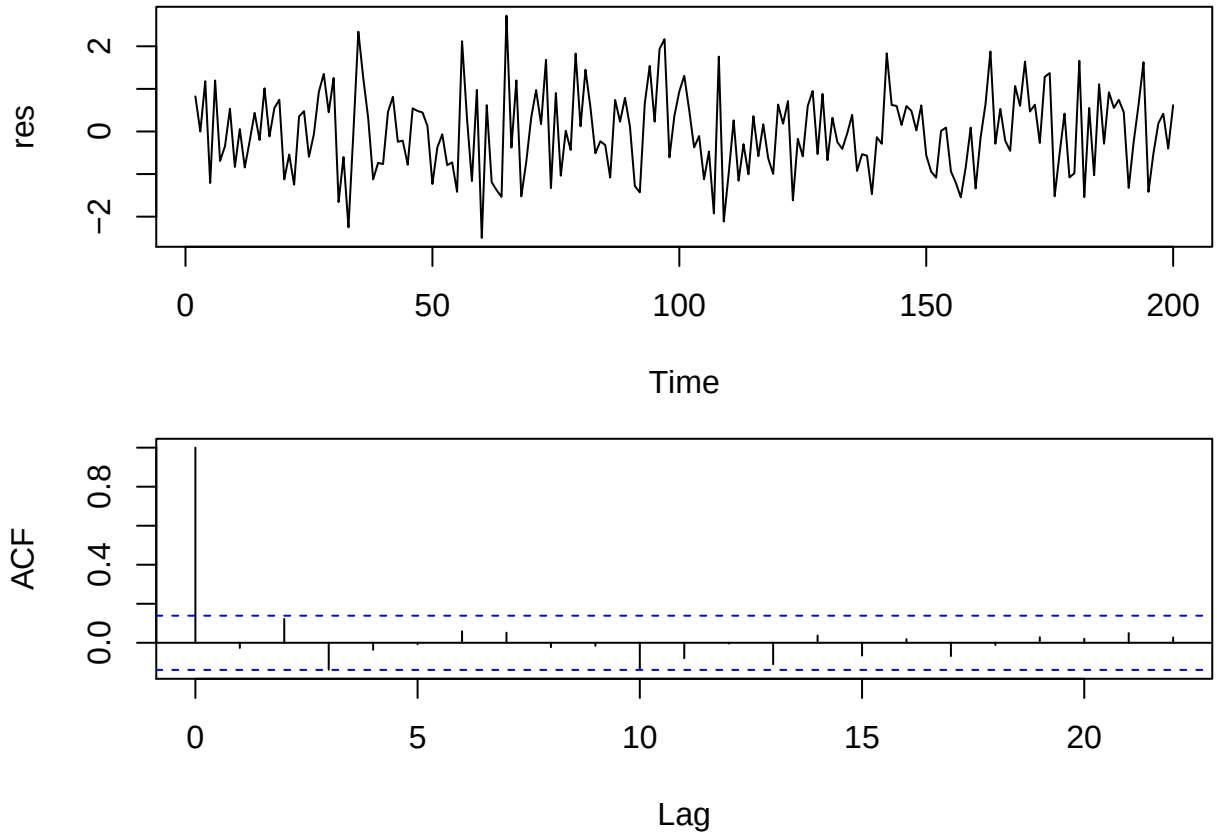
$$\hat{x}_t = \bar{x} + \hat{\alpha}_1 \cdot (x_{t-1} - \bar{x}), \quad t \geq 2.$$

Given the predictions we can estimate the model errors as usual by the model residuals:

$$\hat{w}_t = x_t - \hat{x}_t, \quad t \geq 2.$$

If we believe the model describes the dataset well the residuals should look like a sample of white noise:

```
res <- na.omit(fit$resid)
par(mfrow = c(2,1), mar = c(4,4,1,1))
plot(res)
acf(res)
```



This naturally looks good for this artificial dataset.

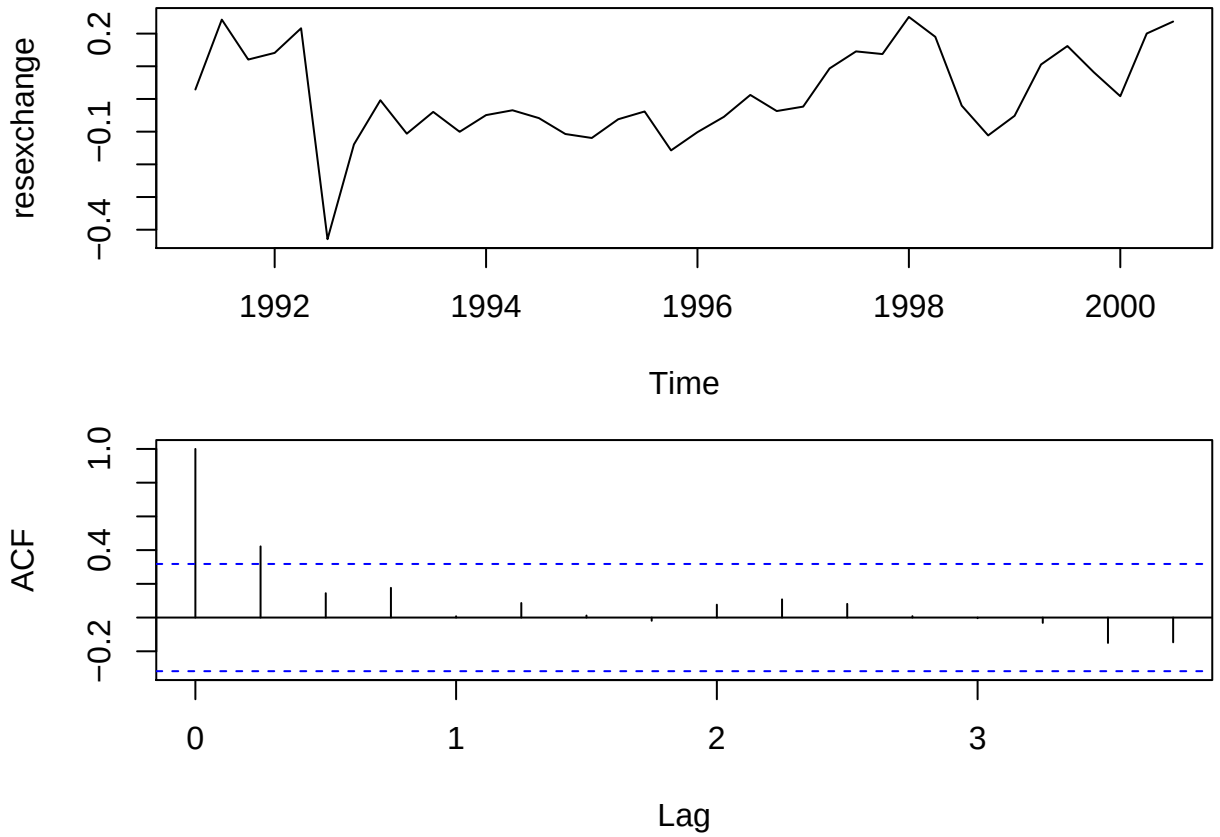
4.1.4 AR(1) model fitted to exchange rate

A random walk is an example of a AR(1) model with $\alpha_1 = 1$, and it is non-stationary. This didn't provide an ideal fit for the exchange rate dataset, so we might suggest a stationary AR(1) model with α_1 as a parameter to be estimated from data:

```
fitexchange <- ar(exchange, order.max = 1)
fitexchange$ar
```

```
## [1] 0.890261
```

```
resexchange <- na.omit(fitexchange$resid)
par(mfrow = c(2,1), mar = c(4,4,1,1))
plot(resexchange)
acf(resexchange)
```



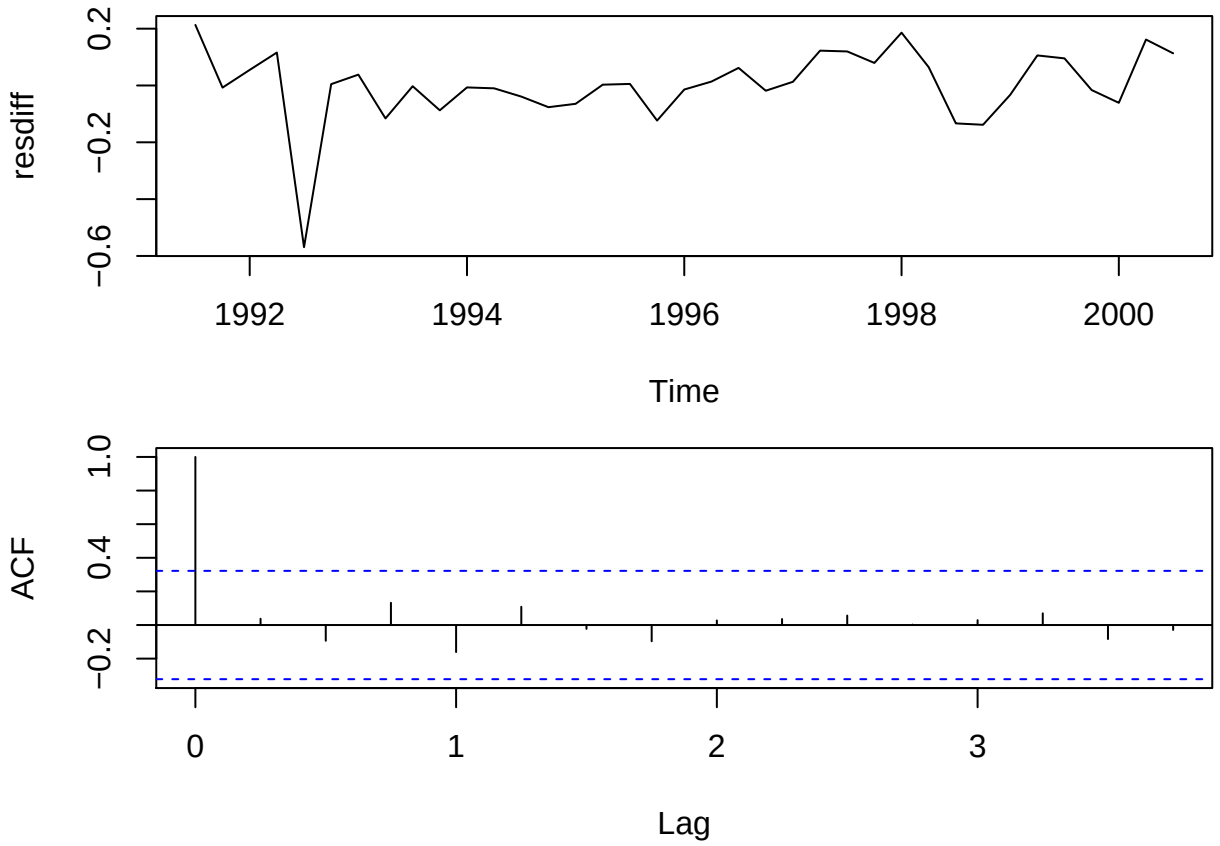
This does not appear to really provide a better fit than the random walk model proposed earlier.

An alternative would be to propose a AR(1) model for the differenced time series $\nabla x_t = x_t - x_{t-1}$:

```
dexchange <- diff(exchange)
fitdiff <- ar(dexchange, order.max = 1)
fitdiff$ar
```

```
## [1] 0.3451507
```

```
resdiff <- na.omit(fitdiff$resid)
par(mfrow = c(2,1), mar = c(4,4,1,1))
plot(resdiff)
acf(resdiff)
```



4.1.5 Prediction from AR(1) model

We can use a fitted AR(1) model to predict future values of a time series. If the last observed time point is t then we predict x_{t+1} using the equation given previously:

$$\hat{x}_{t+1} = \bar{x} + \hat{\alpha}_1 \cdot (x_t - \bar{x}).$$

If we want to predict x_{t+2} we use

$$\hat{x}_{t+2} = \bar{x} + \hat{\alpha}_1 \cdot (\hat{x}_{t+1} - \bar{x}).$$

And we can continue this way. Prediction is performed by `predict` in R. E.g. for the AR(1) model fitted to the exchange rate data the last observation is in third quarter of 2000. If we want to predict 1 year ahead to third quarter of 2001 (probably a bad idea due to the stochastic trend):

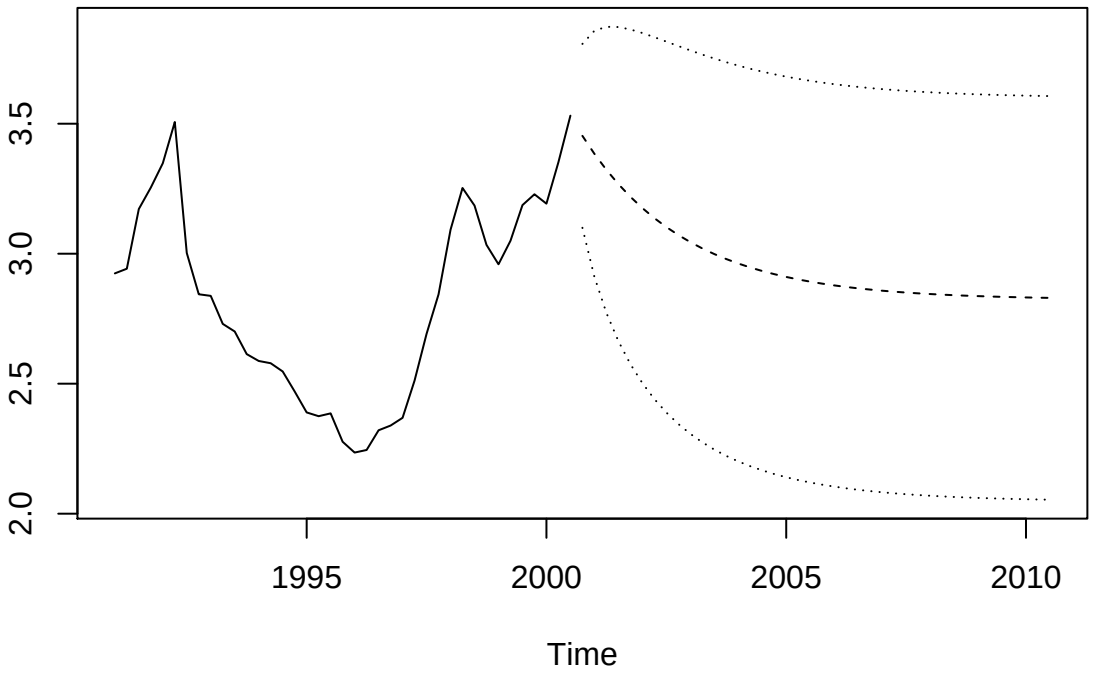
```
pred1 <- predict(fitexchange, n.ahead = 4)
pred1
```

```
## $pred
##          Qtr1     Qtr2     Qtr3     Qtr4
## 2000                           3.453332
## 2001 3.384188 3.322631 3.267830
##
## $se
##           Qtr1      Qtr2      Qtr3      Qtr4
## 2000                               0.1767767
## 2001 0.2366805 0.2750411 0.3020027
```

Note how the prediction returns both the predicted value and a standard error for this value. So we predict that the exchange rate in third quarter of 2001 would be within 3.27 ± 0.6 with approximately 95% probability.

We can plot a prediction and approximate 95% pointwise prediction intervals with `ts.plot` (where we use a 10 year prediction – which is a very bad idea – to see how it behaves in the long run):

```
pred10 <- predict(fitexchange, n.ahead = 40)
lower10 <- pred10$pred-2*pred10$se
upper10 <- pred10$pred+2*pred10$se
ts.plot(exchange, pred10$pred, lower10, upper10, lty = c(1,2,3,3))
```



4.2 Auto-regressive models of higher order

The first order auto-regressive model can be generalised to higher order by adding more lagged terms to explain the current value x_t. An AR(p) process is

$$x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \cdots + \alpha_p x_{t-p} + w_t$$

where w_t is white noise and $\alpha_1, \alpha_2, \ldots, \alpha_p$ are parameters to be estimated from data.

The parameters cannot be chosen arbitrarily if we want the model to be stationary. To check that a given AR(p) model is stationary we must find all the roots of the characteristic equation

$$1 - \alpha_1 z - \alpha_2 z^2 - \cdots - \alpha_p z^p = 0$$

and check that the absolute value of all the roots is greater than 1.

4.2.1 Estimation of AR(p) models

For an AR(p) model there are typically two things we need to estimate:

1. The maximal non-zero lag p in the model.
2. The autoregressive coefficients/parameters $\alpha_1, \ldots, \alpha_p$.

For the first point we note the following theoretical property of AR(p) processes:

For an AR(p) process the theoretical value of the partial autocorrelation at lag k is α_k. Thus, for all lags greater than p the theoretical partial autocorrelation is zero. So heuristically we can choose the maximal lag p of an AR(p) process by looking at when the estimated partial autocorrelation function is close to zero.

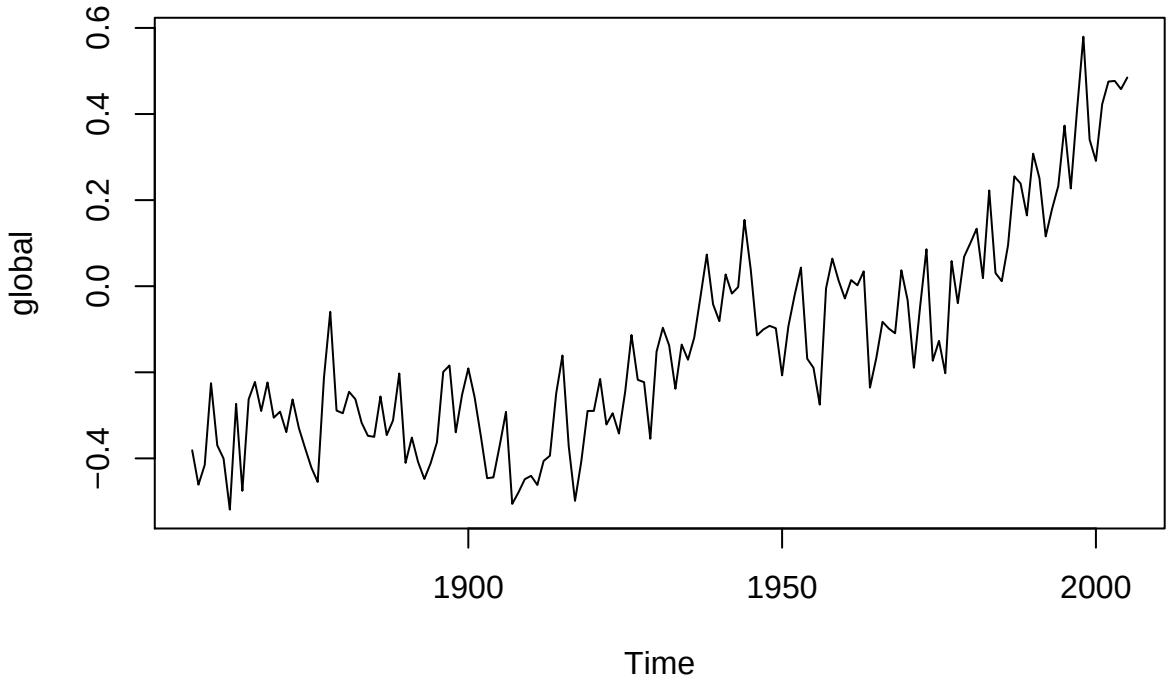
In practice it can be hard to say exactly when the pacf is sufficienty close to zero and more advanced methods are often used. The function `ar` in R uses AIC (Akaike's Information Criterion) to automatically select the value for p.

Once the order is chosen and the estimates $\hat{\alpha}_1, \ldots, \hat{\alpha}_p$ are found the corresponding standard errors can be found as the square root of the diagonal of the matrix stored as `asy.var.coef` in the fitted model object.

4.2.2 Example of AR(p) model

We use an example of monthly global temperatures expressed as anomalies from the monthly average in 1961-1990. We reduce the dataset to the yearly mean temperature, and fit an AR(p) model to this. A good fit would indicate that the higher temperatures over the last decade could be explained by a purely stochastic process which just has dependence on the temperature anomalies from previous year and eventually might as well start decreasing again. **However, this does not mean that there is no climate crisis!** There is lots of scientific evidence of this based on much more complicated models and more detailed data.

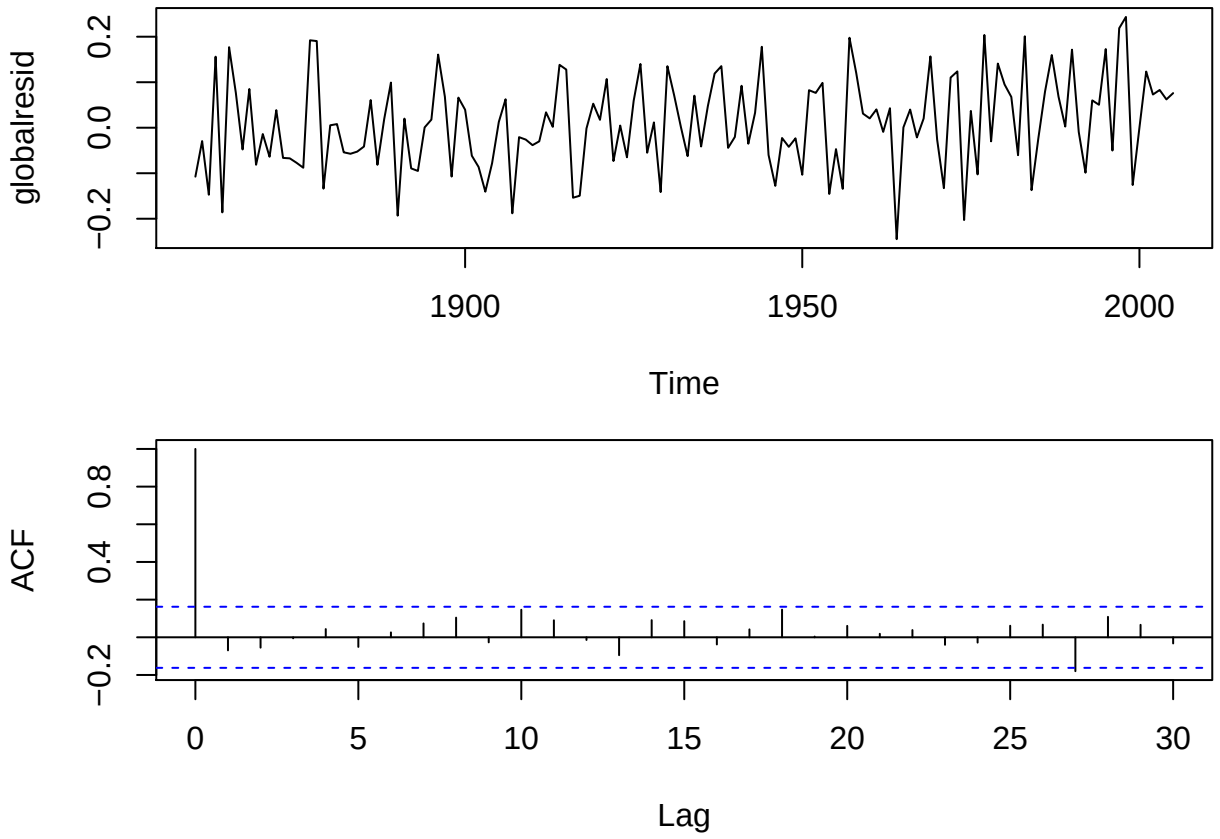
```
global_data <- scan("https://asta.math.aau.dk/eng/static/datasets?file=global.dat")
global_monthly <- ts(global_data, st = c(1856,1), end = c(2005,12), freq = 12)
global <- aggregate(global_monthly, FUN = mean)
plot(global)
```




```
globalfit <- ar(global, order.max = 10)
globalfit
```

```
##
## Call:
## ar(x = global, order.max = 10)
##
## Coefficients:
##      1       2       3       4
## 0.6825  0.0032  0.0672  0.1730
##
## Order selected 4  sigma^2 estimated as  0.01371
```

```
globalresid <- na.omit(globalfit$resid)
par(mfrow = c(2,1), mar = c(4,4,1,1))
plot(globalresid)
acf(globalresid, lag.max = 30)
```



5 Moving average models

Another class of models are moving average (MA) models. An moving average process of order q, MA(q), is defined by

$$x_t = w_t + \beta_1 w_{t-1} + \beta_2 w_{t-2} + \cdots + \beta_q w_{t-q}$$

where w_t is a white noise process with mean zero and variance σ_w^2 and $\beta_1, \beta_2, \ldots, \beta_q$ are parameters to be estimated.

Since a moving average process is a finite sum of stationary white noise terms it is itself stationary and therefore the mean and variance is time-invariant (same constant mean and variance for all t):

- Mean $\mu(t) = 0$
- Variance $\sigma^2(t) = \sigma_w^2(1 + \beta_1^2 + \beta_2^2 + \cdots + \beta_q^2)$

The autocorrelation function, for $k \geq 0$, is

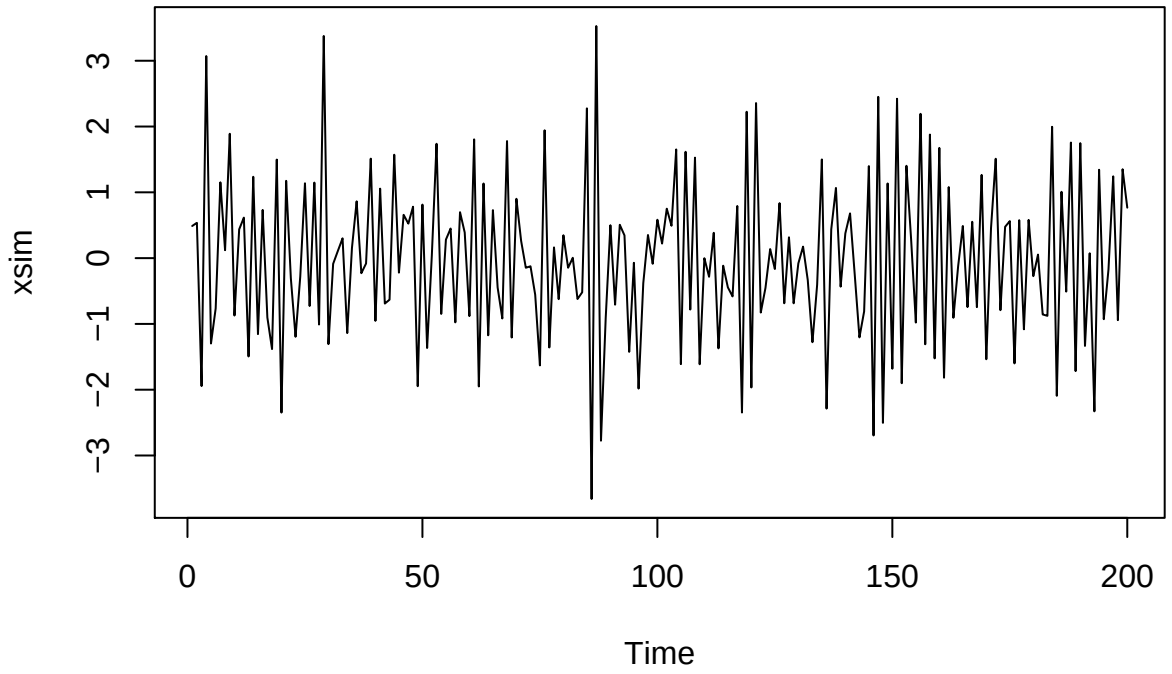
$$\rho(k) = \begin{cases} 1 & k = 0 \\ \sum_{i=0}^{q-k} \beta_i \beta_{i+k} / \sum_{i=0}^{q} \beta_i^2 & k = 1, 2, \ldots, q \\ 0 & k > q \end{cases}$$

where $\beta_0 = 1$.

5.1 Simulation of MA(q) processes

To simulate a MA(q) process we just need the white noise process w_t and then transform it using the MA coefficients. If we e.g. want to simulate a model with $\beta_1 = -0.7$, $\beta_2 = 0.5$, and $\beta_3 = -0.2$ we can use `arima.sim`:

```
xsim <- arima.sim(list(ma = c(-.7, .5, -.2)), n = 200)
plot(xsim)
```



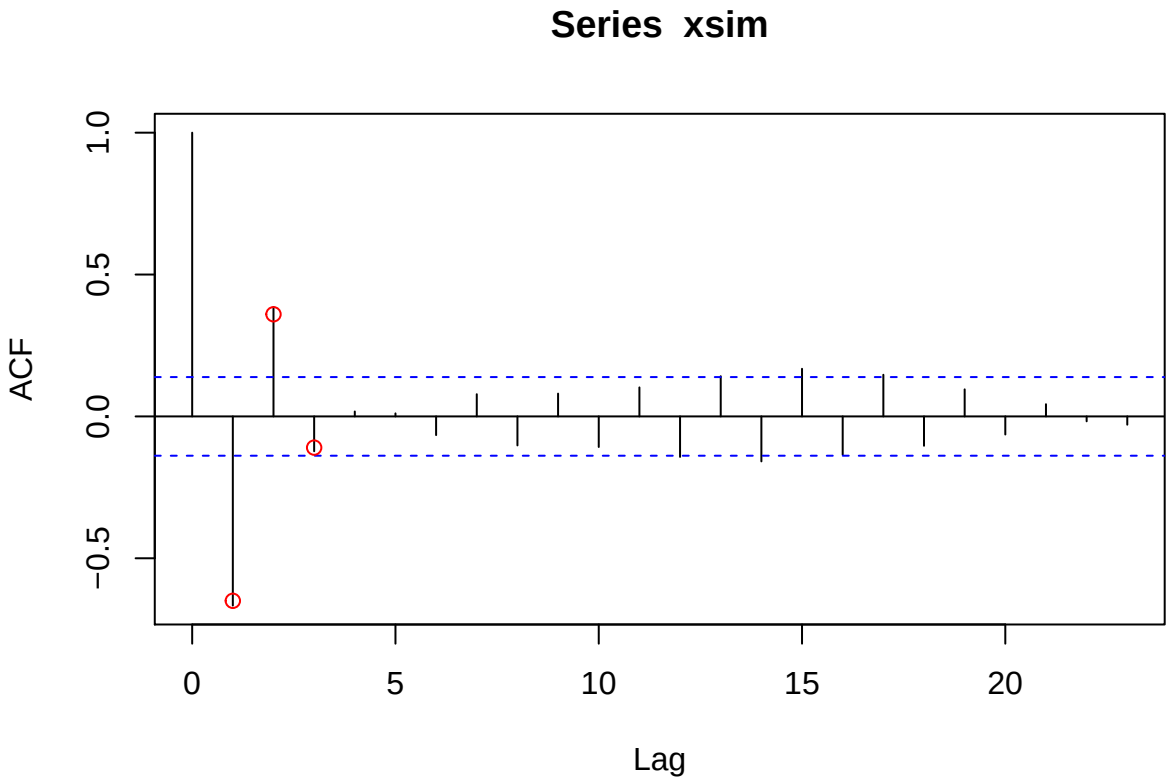
The theoretical autocorrelations are in this case:

$$\rho(1) = \frac{1 \cdot (-0.7) + (-0.7) \cdot 0.5 + 0.5 \cdot (-0.2)}{1 + (-0.7)^2 + 0.5^2 + (-0.2)^2} = -0.65$$

$$\rho(2) = \frac{1 \cdot 0.5 + (-0.7) \cdot (-0.2)}{1 + (-0.7)^2 + 0.5^2 + (-0.2)^2} = 0.36$$

$$\rho(3) = \frac{1 \cdot (-0.2)}{1 + (-0.7)^2 + 0.5^2 + (-0.2)^2} = -0.11$$

```
acf(xsim)
points(1:3, c(-.65, .36, -.11), col = "red")
```



5.1.1 Estimation of MA(q) models

To estimate the parameters of a MA(q) model we use `arima`:

```
xfit <- arima(xsim, order = c(0,0,3))
xfit
```

```
##
## Call:
## arima(x = xsim, order = c(0, 0, 3))
##
## Coefficients:
##           ma1     ma2      ma3  intercept
##       -0.7395  0.5630  -0.1814    -0.0353
## s.e.   0.0710  0.0779   0.0731     0.0415
##
## sigma^2 estimated as 0.8325:  log likelihood = -265.84,  aic = 541.68
```

The function `arima` does not include automatic selection of the order of the model so this has to be chosen beforehand or selected by comparing several proposed models using the AIC.

6 Mixed models: Auto-regressive moving average models

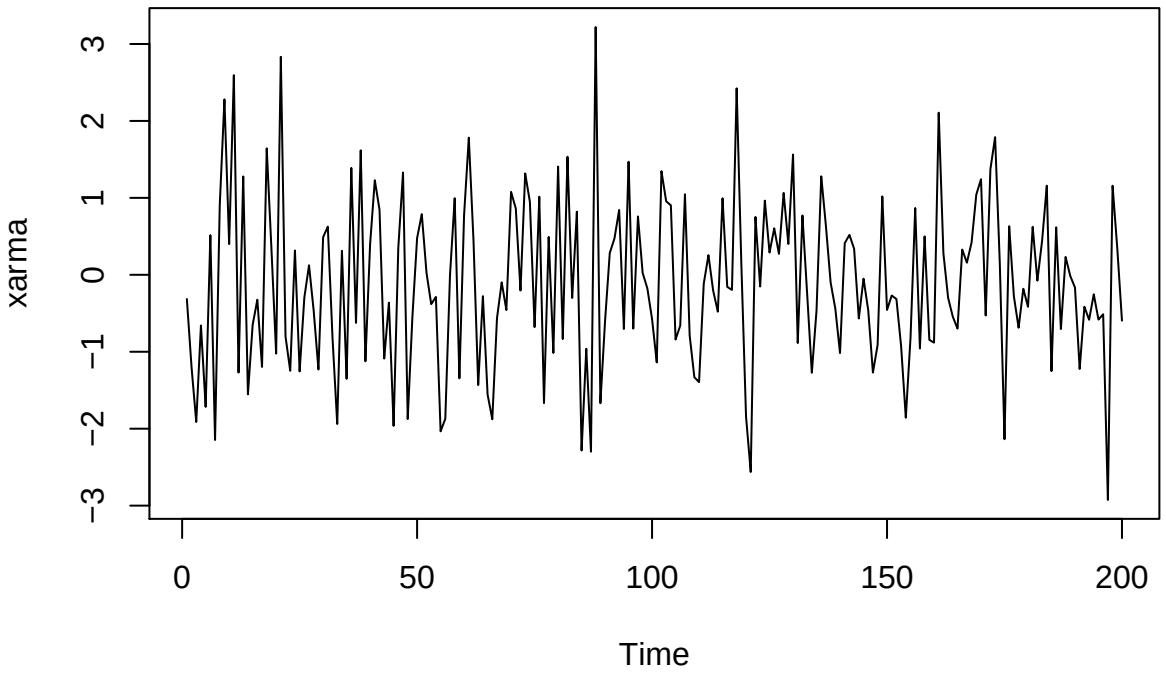
A time series x_t follows an auto-regressive moving average (ARMA) process of order (p, q), denoted $ARMA(p, q)$, if

$$x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \cdots + \alpha_p x_{t-p} + w_t + \beta_1 w_{t-1} + \beta_2 w_{t-2} + \cdots + \beta_q w_{t-q}$$

where w_t is a white noise process and $\alpha_1, \alpha_2, \ldots, \alpha_p, \beta_1, \beta_2, \ldots, \beta_q$ are parameters to be estimated.

We can simulate an ARMA model with `arima.sim`. E.g. an ARMA(1,1) model:

```
xarma <- arima.sim(model = list(ar = -0.6, ma = 0.5), n = 200)
plot(xarma)
```



Estimation is done with `arima` as before.

6.0.1 Example with exchange rate data

For the exchange rate data we may e.g. suggest either a AR(1), MA(1) or ARMA(1,1) model. We can compare fitted model using `AIC` (smaller is better):

```
exchange_ar <- arima(exchange, order = c(1,0,0))
AIC(exchange_ar)
```

```
## [1] -37.40417
```

```
exchange_ma <- arima(exchange, order = c(0,0,1))
AIC(exchange_ma)
```

```
## [1] -3.526895
```

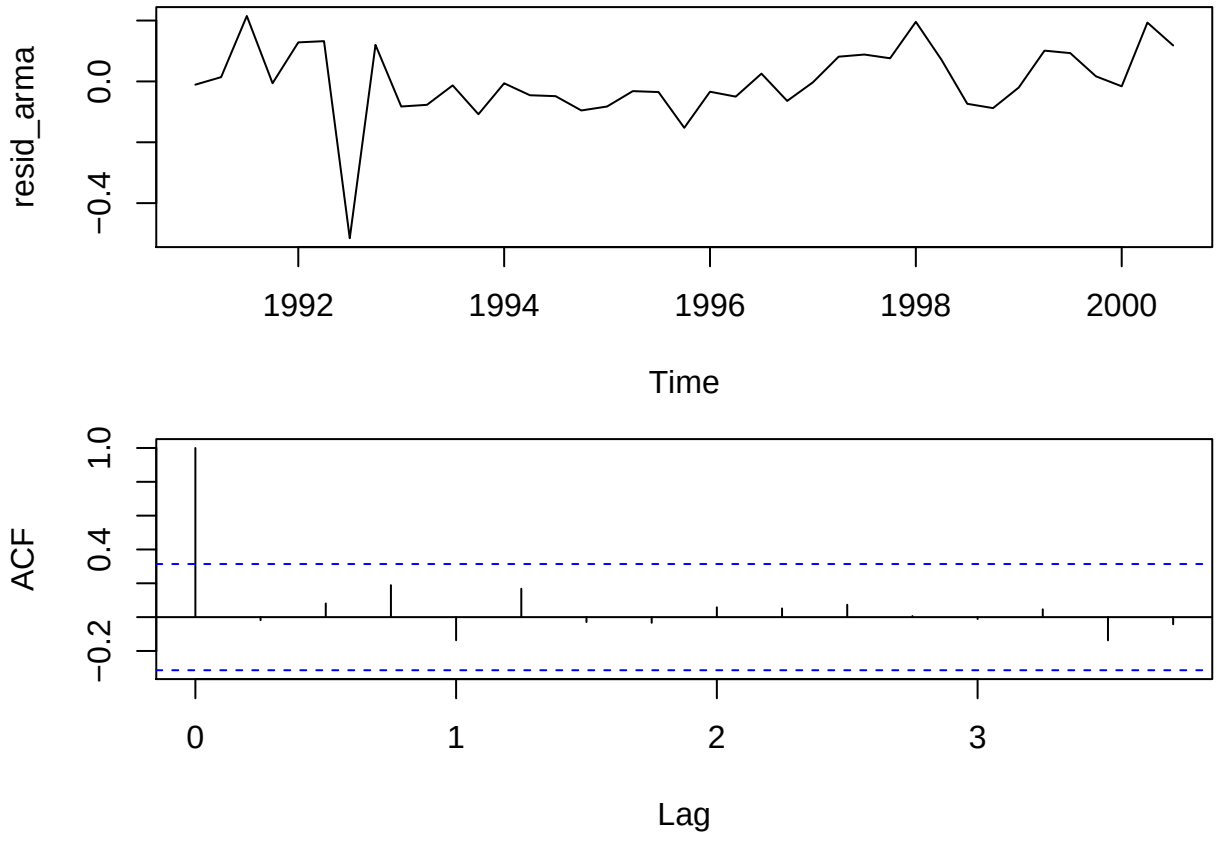
```
exchange_arma <- arima(exchange, order = c(1,0,1))
AIC(exchange_arma)
```

```
## [1] -42.27357
```

```
exchange_arma
```

```
##
## Call:
## arima(x = exchange, order = c(1, 0, 1))
##
## Coefficients:
##          ar1     ma1  intercept
##       0.8925  0.5319     2.9597
## s.e.  0.0759  0.2021     0.2435
##
## sigma^2 estimated as 0.01505:  log likelihood = 25.14,  aic = -42.27
```

```
par(mfrow = c(2,1), mar = c(4,4,1,1))
resid_arma <- na.omit(exchange_arma$residuals)
plot(resid_arma)
acf(resid_arma)
```



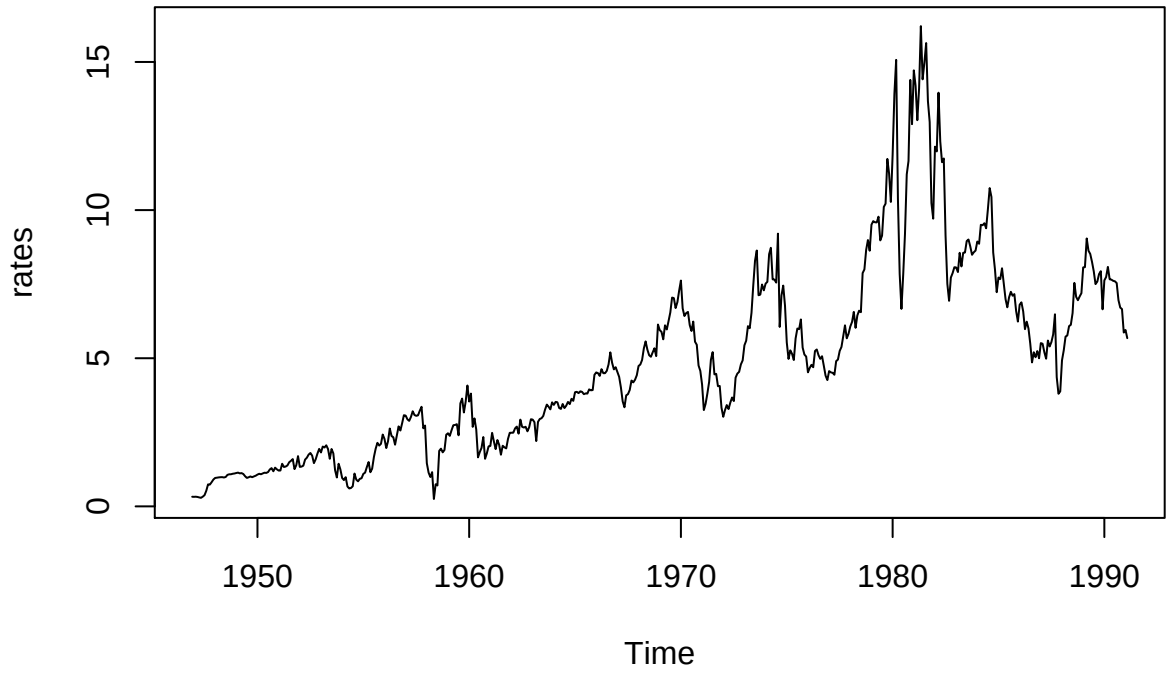
7 Continuous time stochastic processes

7.1 Data example

In this lecture we will study a type of data that on the surface looks like data from time series analysis as presented in the previous lectures. As an illustration, we shall study the data set `Irates`, that contains information about interest rates in the U.S. between 1946 and 1991.

We are interested in the relation between the two variables:

- `t=time`: The time point of the measurement.
- `X_t=rate`: The interest rate at the corresponding time point.



The plot should be understood as follows: For each time point (between 1946 and 1991) there is a value of the interest rate called X_t. So X_t could be seen as a function of t, and this function is plotted.

In principle we imagine that there are infinitely many data points, simply because there are infinitely many time points between 1946 and 1991. Of course this is never true: In practice we will always only have finitely many data points.

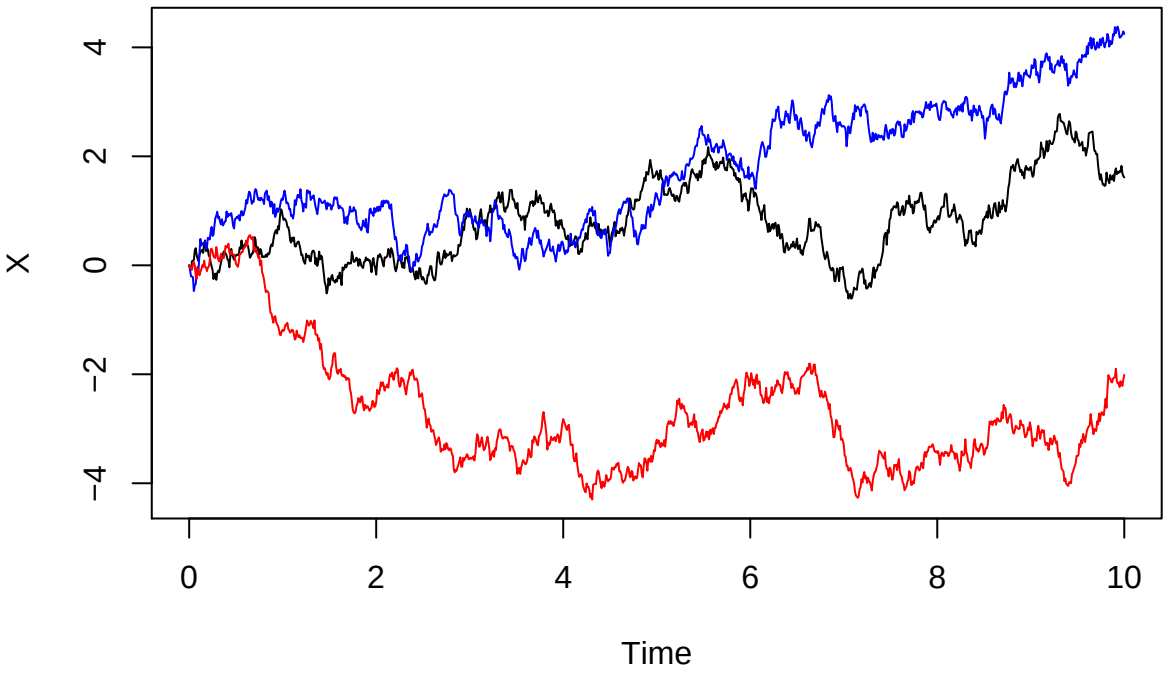
But it makes sense to believe that the real data actually contains all the data points. We are just not able to measure them (and to store them in a computer).

With a model for all datapoints, we are - through simulation - able to describe the behaviour of data. Also between the observations.

8 Wiener process

A key example of a process in continuous time will be the so–called Wiener process.

Three simulated realizations (black, blue and red) of this process can be seen here



A Wiener process has the following properties:

- It starts in 0: $W_0 = 0$.
- It has independent increments: For $0 < s < t$ it holds that $W_t - W_s$ is independent of everything that has happened up to time s, that is W_u for all $u \leq s$.
- It has normally distributed increments: For $0 < s < t$ it holds that the increment $W_t - W_s$ is normally distributed with variance $t - s$:

$$W_t - W_s \sim N(\mu = 0, \sigma^2 = t - s).$$

The intuition of this process is that it somehow changes direction all the time: How the process changes after time s will be independent of what has happened before time s. So whether the process should increase or decrease after s will not be affected by how much it was increasing or decreasing before.

This gives the very bumpy behaviour over time.

9 Differential equation models

9.1 Ordinary differential equations

A common way to define a continuous time stochastic process model is through a stochastic differential equation (SDE) which we will turn to shortly, but before doing so we will recall some basic things about ordinary differential equations.

Suppose f is a differentiable function. Recall the mathematical description of a differential equation

$$\frac{df(t)}{dt} = -4f(t)$$

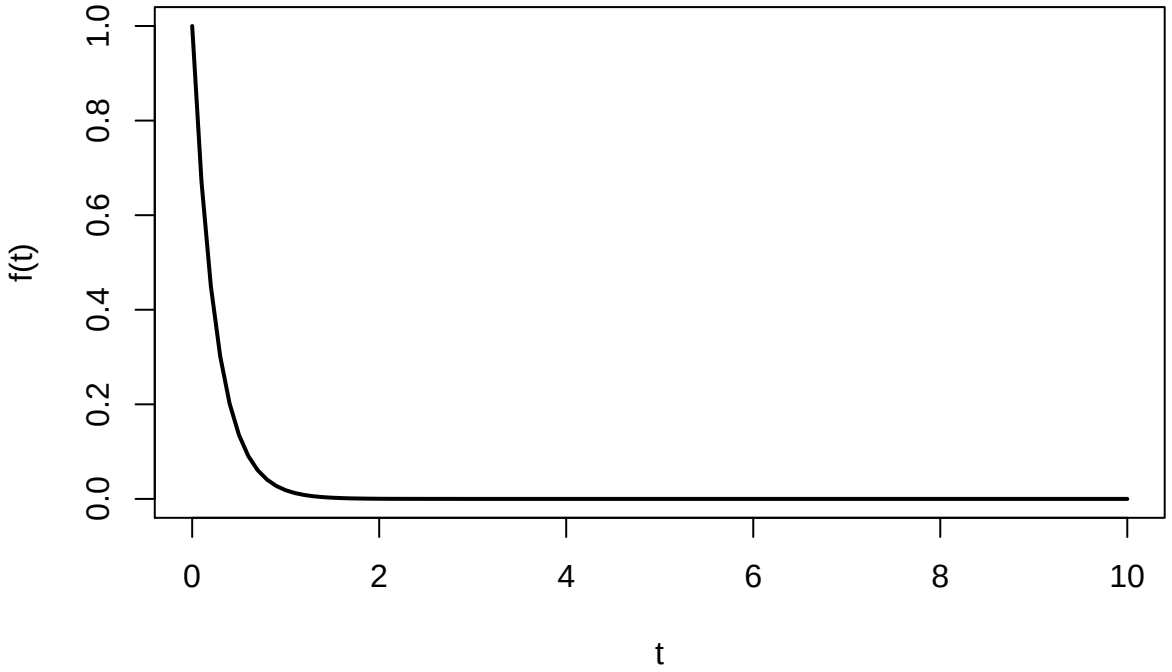
This equation has the solution

$$f(t) = C \cdot \exp(-4t)$$

for any constant C. If we furthermore know that $f(0) = 1$, then $C = 1$ and

$$f(t) = \exp(-4t)$$

The solution can be seen below



With a slightly unusual notation we can rewrite this as

$$df(t) = -4 \cdot f(t)dt$$

This equation has the following (hopefully intuitive) interpretation:

- We imagine that we increase the time point from t to $t + dt$, where dt is something small. So the time is increased by dt.
- Then the value of f is (approximately) changed from $f(t)$ to $f(t) - 4f(t)dt$. So actually the value of f is decreased by $4f(t)\,dt$

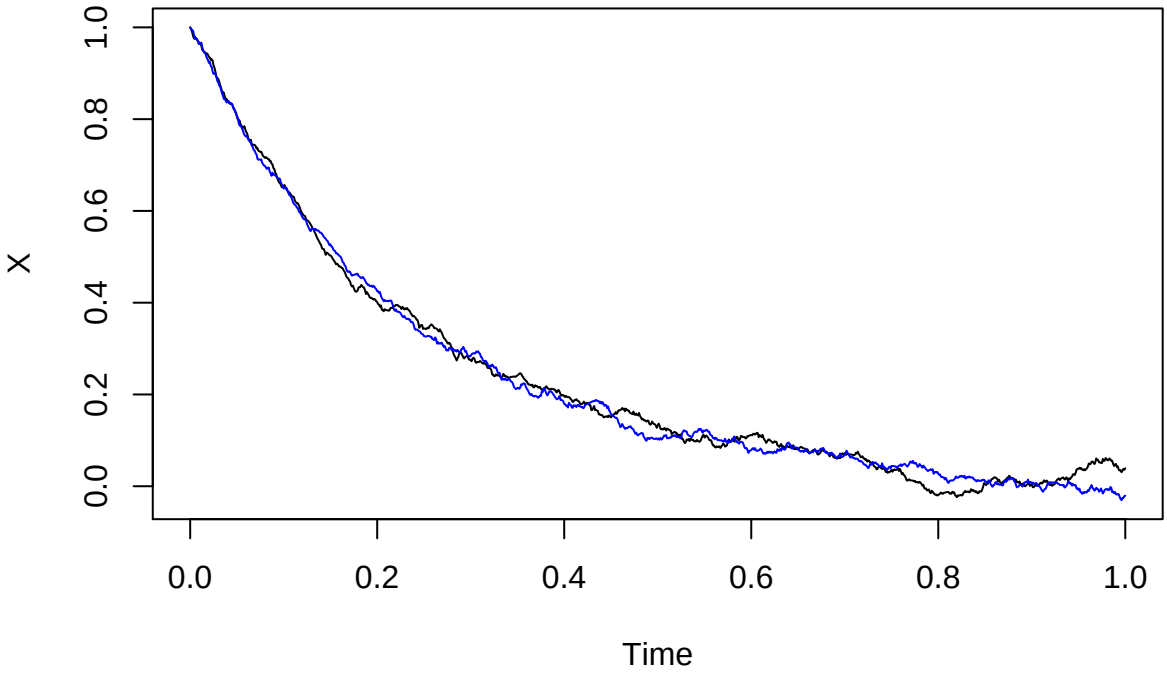
So when t is increased, then $f(t)$ is decreased. And the decrease is determined by the value of $f(t)$. That is why f decreases slower and slower, when t is increased.

We say that the function has a **drift** towards zero, and this drift is determined by the value of the function.

9.2 Stochastic differential equations

It will probably never be true that data behaves exactly like the exponentially decreasing curve on the previous slide.

Instead we will consider a model, where some random noise from a Wiener process has been added. Two different (black/blue) simulated realizations can be seen below



The type of process that is simulated above is most often described formally by the equation

$$dX_t = -4X_t dt + 0.1 dW_t$$

This is called a **Stochastic Differential Equation** (SDE), and the processes simulated above are called solutions of the stochastic differential equation.

The SDE $dX_t = -4X_t dt + 0.1 dW_t$ has two terms:

- $-4X_t dt$ is the **drift term**.
- $0.1 dW_t$ is the **diffusion term**.

The intuition behind this notation is very similar to the intuition in the equation $df(t) = -4 \cdot f(t)\, dt$ for an ordinary differential equation. When the time is increased by the small amount dt, then the process X_t is increased by $-4X_t\, dt$ AND by how much the process $0.1W_t$ has increased on the time interval $[t, t + dt]$.

So this process has a **drift** towards zero, but it is also pushed in a random direction (either up or down) by the Wiener process (more precisely, the process $0.1W_t$)

9.2.1 Simulation examples

Firstly we simulate the SDE from before

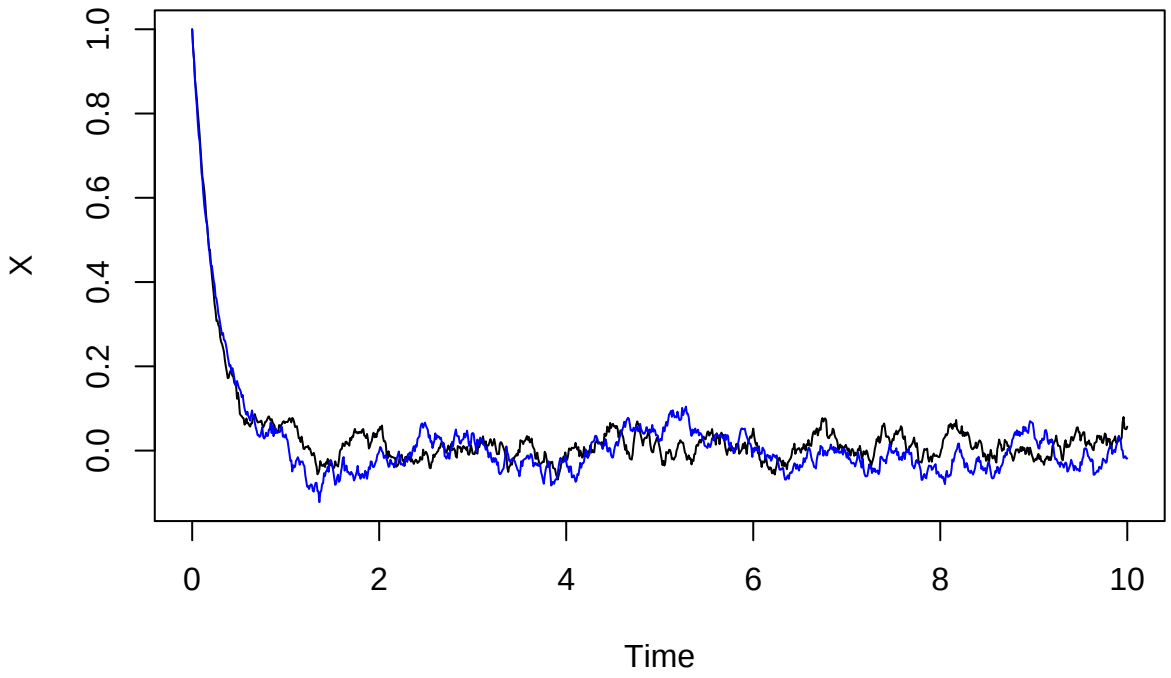
$$dX_t = -4X_t dt + 0.1 dW_t$$

For this, we need the package `Sim.DiffProc`. We use the function `snssde1` for which we have to specify the drift term and the diffusion term as R-functions of x. In this case, the function for the diffusion term is constantly equal to 0.1.

The parameters needed in the function input are:

- `drift` is the function determining the drift term.
- `diffusion` is the function determining the diffusion term.
- `M` is the desired number of realizations of the process.
- `N` is the number of simulation steps (`R` does not simulate a continuous curve but a lot of connected dots, and this is the number of dots).
- `t0` is the initial time of the simulated process.
- `T` is the ending time of the simulated process.
- `x0` is the initial value of the process (the value of X at time `t0`).

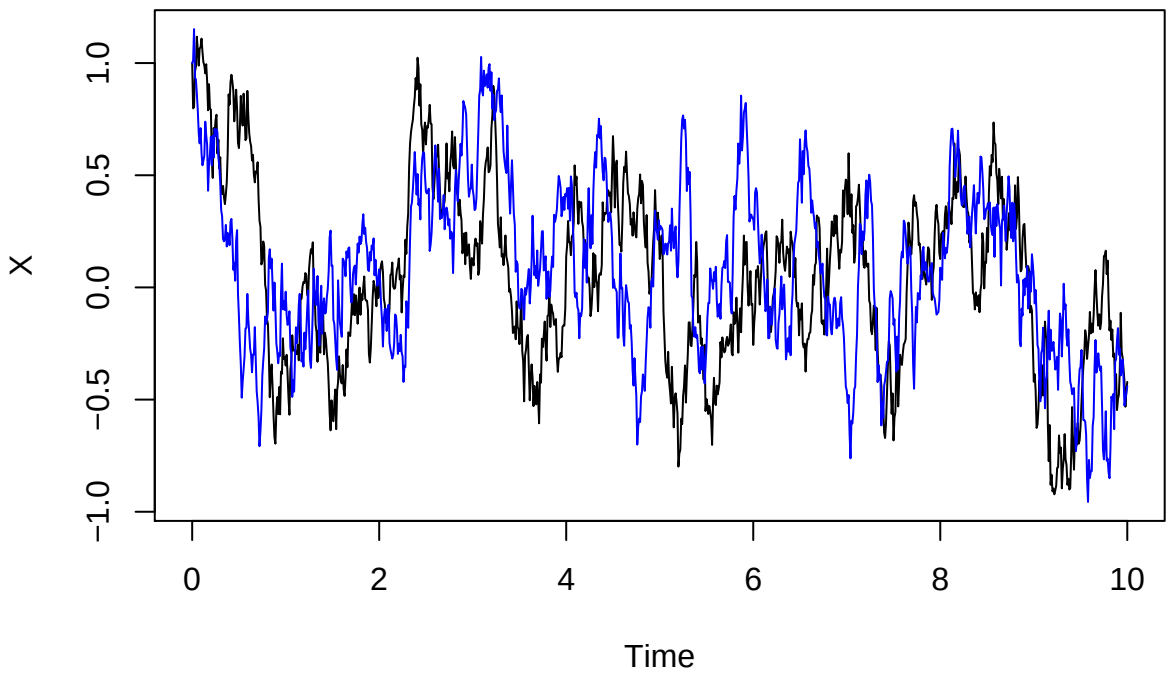
```
library(Sim.DiffProc)
f <- expression(-4*x)    ## the drift term as a function of x
g <- expression(0.1)      ## the diffusion term as a function of x
res <- snssde1d(drift=f, diffusion=g, M=2, N=1000, t0=0, T=10, x0=1)
plot(res, plot.type = 'single', col = c('black','blue'))
```



With the ending time being larger than before, we see that the process stabilizes around 0: There is a drift towards 0, but also some noise pushing the process away from 0.

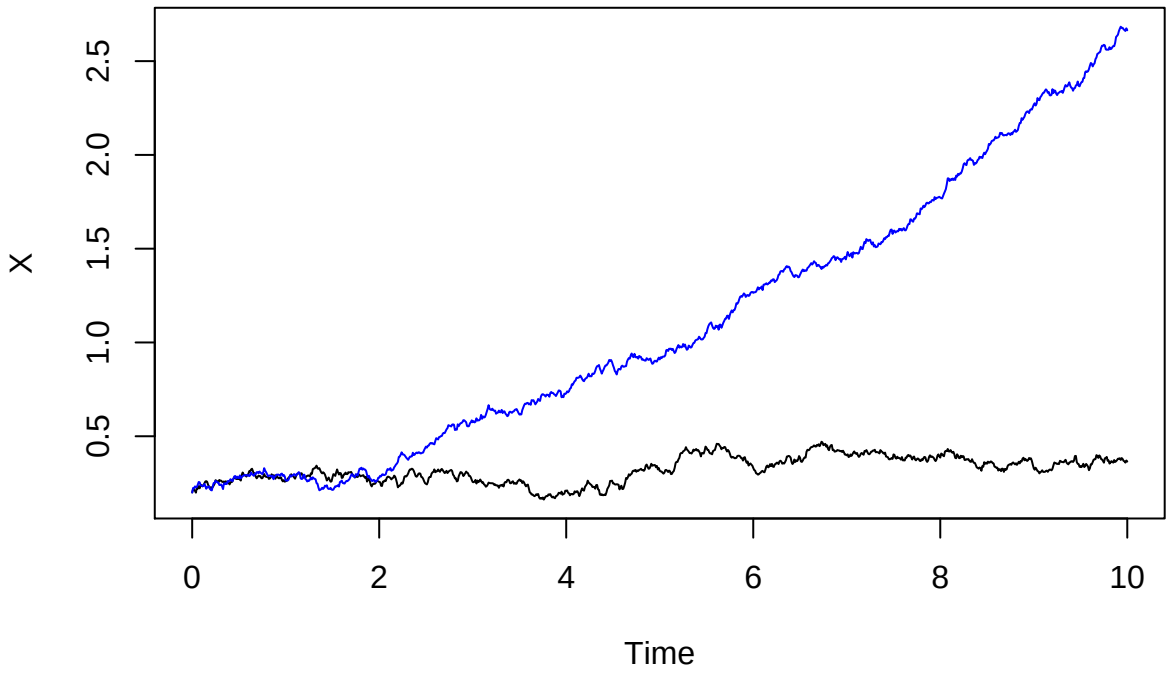
Increasing the diffusion parameter from 0.1 to 1 (i.e. $dX_t = -4X_t dt + 1dW_t$) makes the process more varying:

```
f <- expression(-4*x)
g <-expression(1)
res <- snssde1d(drift=f, diffusion=g, M=2, N=1000, t0=0, T=10, x0=1)
plot(res, plot.type='single', col=c('black','blue'))
```



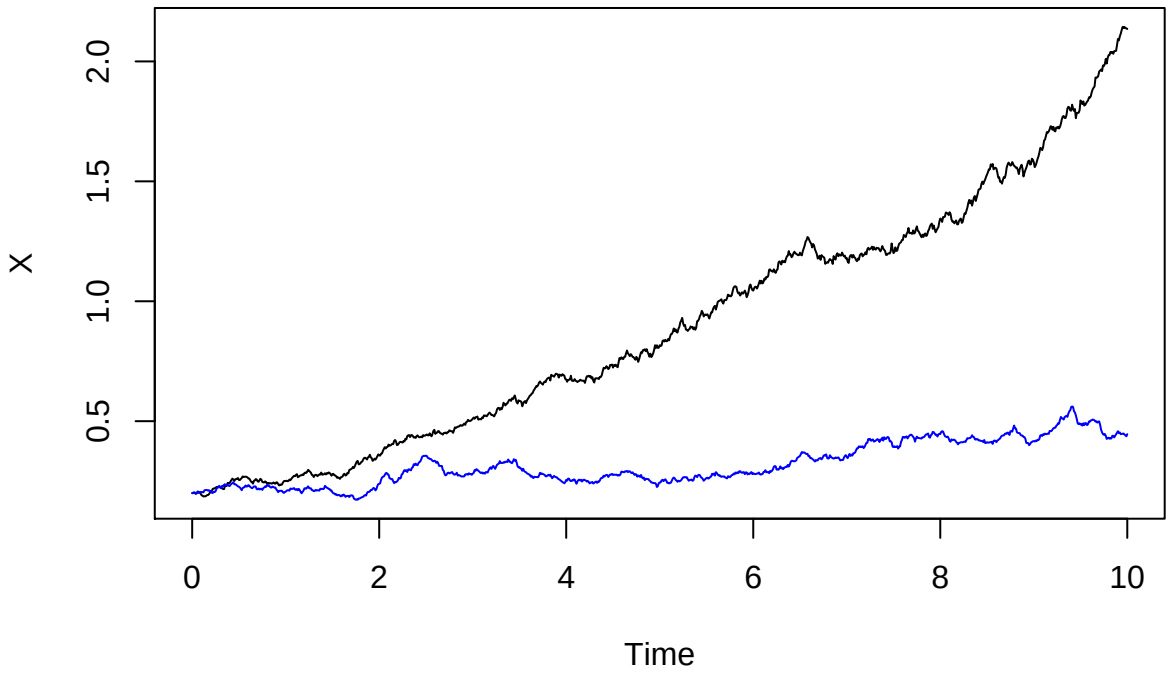
Making the drift parameter positive (e.g. $dX_t = 0.2X_t dt + 0.1 dW_t$) drives the process away from 0:

```
f <- expression(0.2*x)
g <-expression(0.1)
res <- snssde1d(drift=f, diffusion=g, M=2, N=1000, t0=0, T=10, x0=0.2)
plot(res,plot.type='single',col=c('black','blue'))
```



Sometimes the noise depends on the value of the process itself. In this case the diffusion term includes X_t. E.g. $dX_t = 0.2X_t dt + 0.1\sqrt{X_t}dW_t$:

```
f <- expression(0.2*x)
g <- expression(0.1*sqrt(x))
res <- snssde1d(drift=f,diffusion=g,M=2,N=1000,t0=0,T=10,x0=0.2)
plot(res,plot.type='single',col=c('black','blue'))
```



9.3 Fitting SDE models to data

When we want to fit a model to data, we will work with the following more general stochastic differential equation

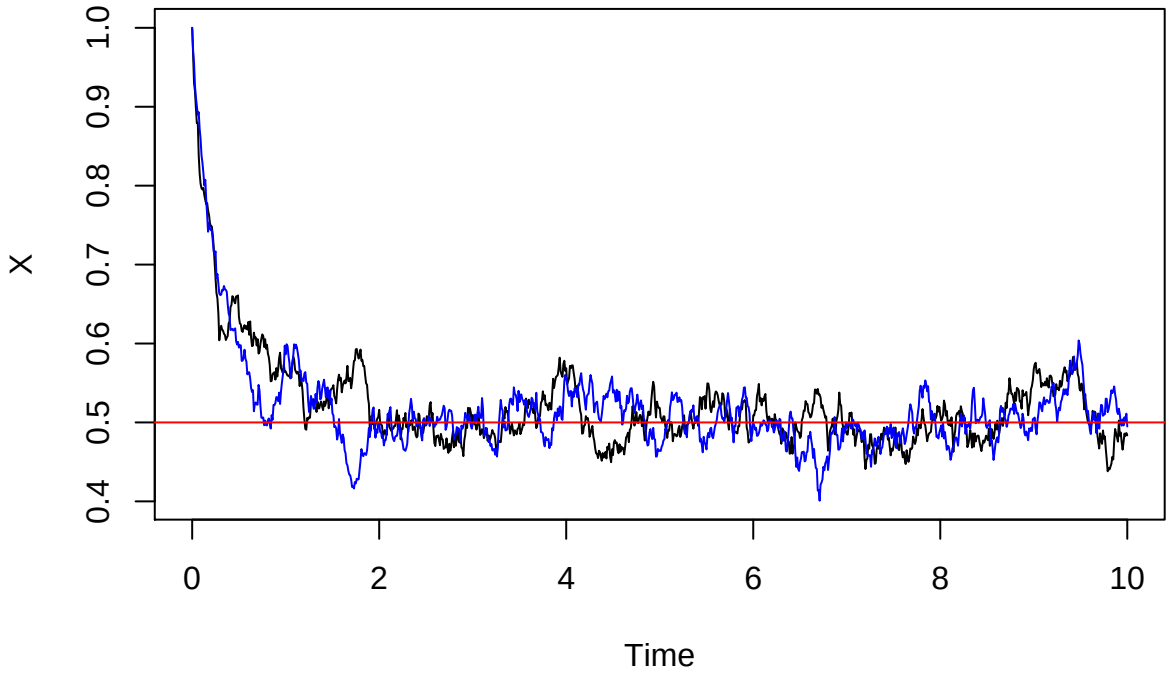
$$dX_t = (\theta_1 + \theta_2 X_t)dt + \theta_3 X_t^{\theta_4} dW_t$$

where $\theta_1, \theta_2, \theta_3, \theta_4$ are parameters, such that $\theta_3, \theta_4 \geq 0$. For a given dataset the goal is then to find (estimate) parameter values such that the model describes the data as well as possible.

We note that:

- The SDE above given by $dX_t = -4X_t dt + 0.1 dW_t$ is the special case with $\theta_1 = 0$, $\theta_2 = -4$, $\theta_3 = 0.1$ and $\theta_4 = 0$ (recall the mathematical convention that $x^0 = 1$).
- The Wiener process $X_t = W_t$ is the special case with $\theta_1 = 0$, $\theta_2 = 0$, $\theta_3 = 1$ and $\theta_4 = 0$.
- The ordinary differential equation is the special case with $\theta_1 = 0$, $\theta_2 = -4$, $\theta_3 = 0$ and $\theta_4 = 0$ (in principle, θ_4 could be anything, when $\theta_3 = 0$).
- The parameters θ_1 and θ_2 control the drift, and in fact it can be shown that the process will drift towards $-\frac{\theta_1}{\theta_2}$ if this is positive, and otherwise it will drift away from this. For example if $dX_t = (2 - 4X_t)dt + 0.1 dW_t$ then X_t will drift towards $-\frac{2}{-4} = 0.5$:

```
f <- expression(2-4*x)
g <- expression(0.1)
res <- snssde1d(drift=f, diffusion=g, M=2, N=1000, t0=0, T=10, x0=1)
plot(res, plot.type='single', col=c('black','blue'))
abline(h = 0.5, col = "red")
```



9.3.1 Fitting an SDE model to interest rate data

Recall the data for U.S. interest rates between 1946 and 1991.

This could look like a stochastic differential equation with $\theta_1 = 0$ and θ_4 being positive, since the process varies more, when it has high values. We can use the function `fitsde` to find the best choice of parameters. Note that in the function we have to give a (good) guess on the parameters (here we use $\theta_1 = 0$, $\theta_2 = 0$, $\theta_3 = 0.5$ and $\theta_4 = 0.5$)

```
data(Irates)
rate <- Irates[ , "r1"]
fx <- expression(theta[1]+theta[2]*x)
## the drift term as a function of x
gx <- expression(theta[3]*x^theta[4])
## the diffusion term as a function of x
fitmod <- fitsde(rate, drift = fx, diffusion = gx,
                 start = list(theta1=0, theta2=0, theta3=0.5, theta4=0.5))
summary(fitmod)
```

```
## Pseudo maximum likelihood estimation
##
## Method:  Euler
## Call:
## fitsde(data = rate, drift = fx, diffusion = gx, start = list(theta1 = 0,
##     theta2 = 0, theta3 = 0.5, theta4 = 0.5))
##
## Coefficients:
##          Estimate Std. Error
## theta1  0.8862133 0.24588655
## theta2 -0.1591198 0.08044809
## theta3  0.7138713 0.03379098
## theta4  0.5926193 0.02765048
##
## -2 log L: 648.049
```

Thus, the estimated model is

$$dX_t = (0.89 - 0.16X_t)dt + 0.71X_t^{0.59}dW_t.$$

We can have the confidence intervals by

```
confint(fitmod)
```

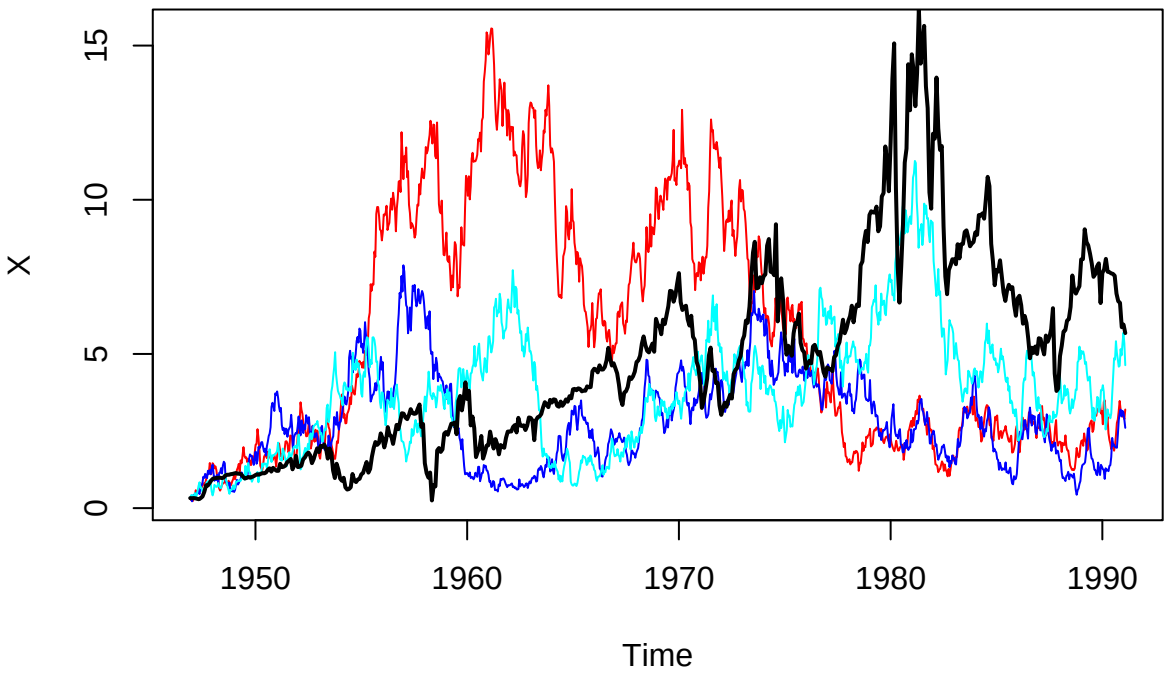
```
##             2.5 %       97.5 %
## theta1  0.4042845  1.368142087
## theta2 -0.3167952 -0.001444485
## theta3  0.6476422  0.780100349
## theta4  0.5384254  0.646813280
```

To draw random realizations from the fitted model we just have to extract the fitted parameters and then use `snssde1` as before:

```
theta <- coef(fitmod)
t <- time(Irates)
s <- snssde1d(drift=fx, diffusion = gx, M = 3, t0 = min(t), T = max(t), x0 = 0.325)
```

A plot of three realizations overlayed the original data:

```
plot(s, plot.type = "single", col = c("red", "blue", "cyan"))
lines(Irates[,'r1'], col = "black", lwd = 2)
```



9.4 Comparing fitted SDE models

If we believe that the data can be better (or equally well) described by another model we can compare the model using the AIC as we did previously for discrete processes.

If we propose a model with no drift (i.e. $\theta_1 = 0$ and $\theta_2 = 0$) we get the following fitted model:

```
f2 <- expression(0)
## No drift term
g2 <- expression(theta[1]*x^theta[2])
## the diffusion term as a function of x
fitmod2 <- fitsde(rate, drift = f2, diffusion = g2,
                  start = list(theta1=0.5, theta2=0.5))
summary(fitmod2)
```

```
## Pseudo maximum likelihood estimation
##
## Method:  Euler
## Call:
```

```
## fitsde(data = rate, drift = f2, diffusion = g2, start = list(theta1 = 0.5,
##     theta2 = 0.5))
##
## Coefficients:
##         Estimate Std. Error
## theta1 0.7400495 0.03454472
## theta2 0.5743530 0.02698523
##
## -2 log L: 661.0032
```

The AIC of the original model with four parameters is lower than the AIC of this new model so we prefer the original model:

```
AIC(fitmod)
```

```
## [1] 656.049
```

```
AIC(fitmod2)
```

```
## [1] 665.0032
```