University of Glasgow

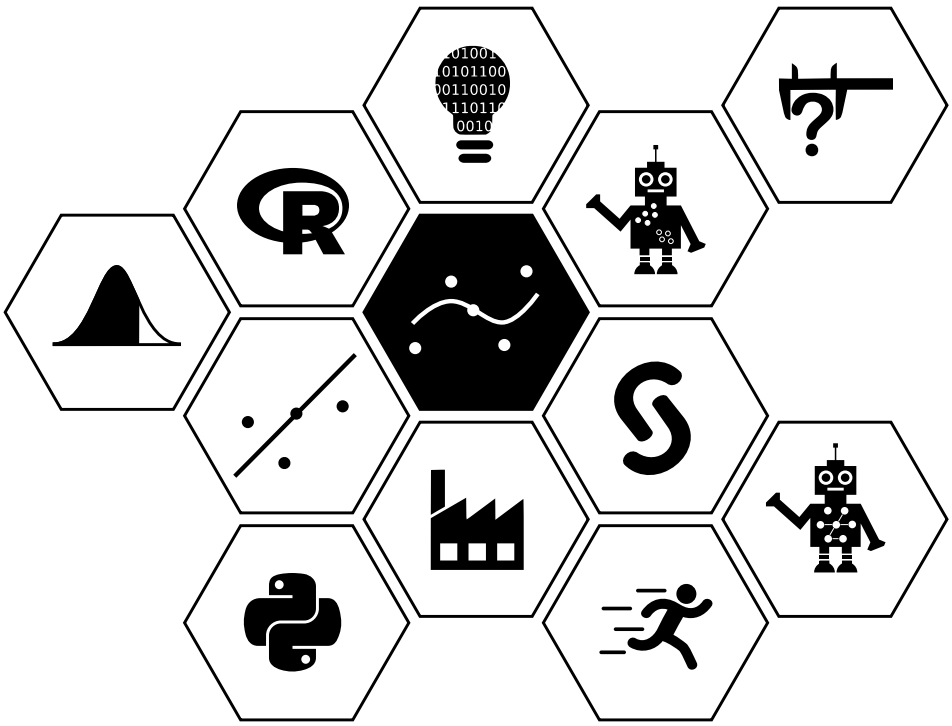
Advanced Predictive Models

Tereza Neocleous

Academic Year 2019-20

Week 8:

ARMA and ARIMA processes; Forecasting in time series



DATA ANALYTICS GLASGOW

Introduction

Last week we introduced the two main classes of time series processes for stationary time series data, autoregressive and moving average processes. This week we will introduce an autoregressive moving average (ARMA) process that may be appropriate if neither an AR nor an MA process succeeds in removing the short-term correlation. Then we will introduce the autoregressive integrated moving average (ARIMA) approach for non-stationary time series data. Finally, we will learn how to forecast (predict) the value of a time series at future points in time.

More general time series processes

In Week 7 we discussed the two most important models for stationary time series data, autoregressive and moving average processes. For most data sets these models will be an adequate representation of the short-term correlation, with autoregressive correlation occurring more often than moving average. However, occasionally we may meet data that are not well represented by either of these time series processes, an example of which is shown below.

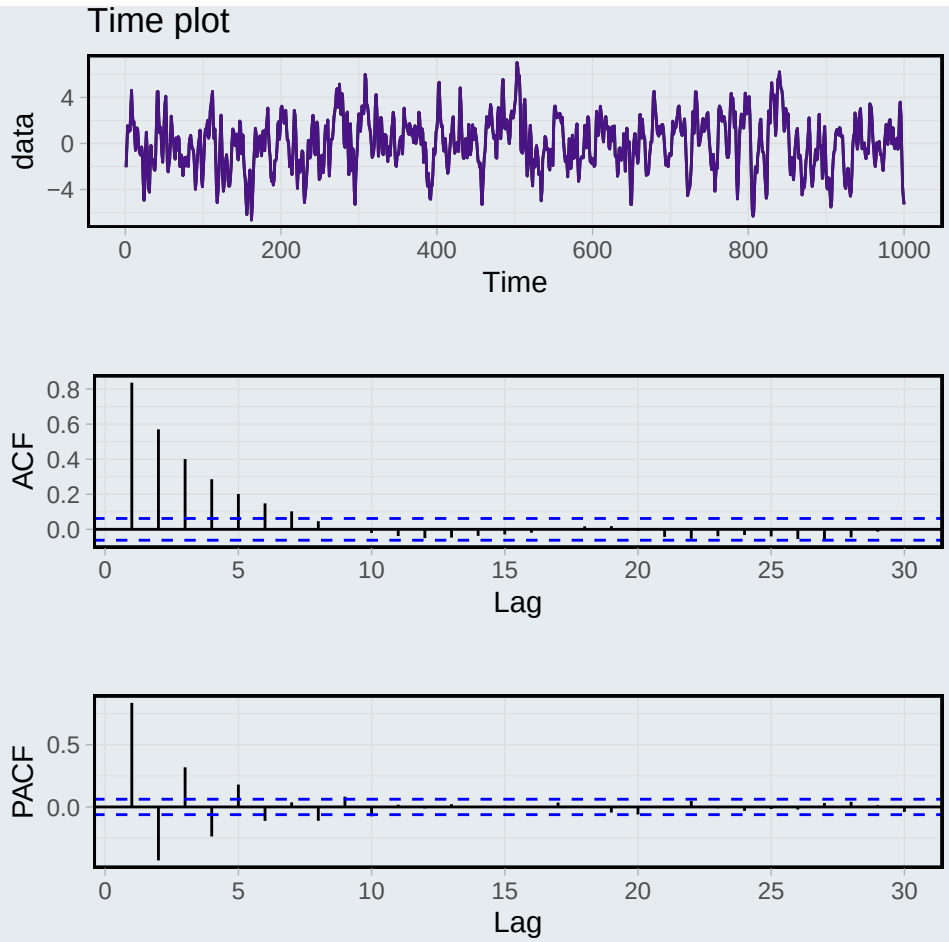
Autoregressive moving average processes

https://youtu.be/IFLsjNONAgw

Duration: 10m25s

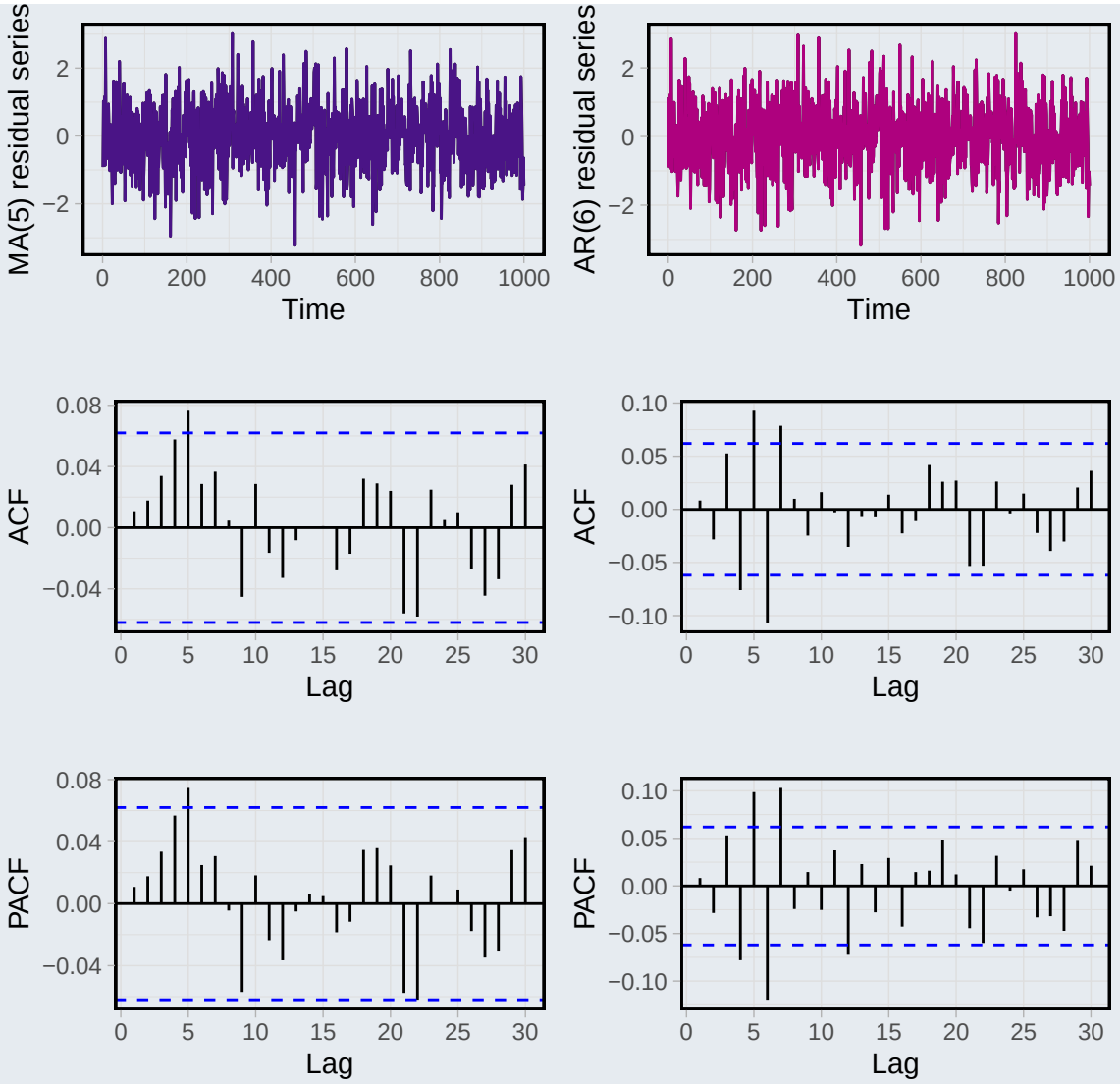
Example 1.

Consider the following data which appear to be stationary but contain short-term correlation.



The ACF and PACF suggest that neither an AR not an MA process is appropriate, but as these are the only models we know, we fit them to the data to see how well they remove the short-term correlation. We chose the order (p and q) as the lowest values that removed the majority of the correlation, which resulted in an AR(6) model or an MA(5) model as shown below.

```
## Fit AR and MA models to the data
model.ma <- arima(data, order=c(0,0,5))
model.ar <- arima(data, order=c(6,0,0))

# Plotting
p1 <- autoplot(model.ma$residuals, main="", ylab="MA(5) residual series") +
      geom_line(colour="#4a1486")
p2 <- autoplot(model.ar$residuals, main="", ylab="AR(6) residual series") +
      geom_line(colour="#ae017e")
p1acf <- autoplot(acf(model.ma$residuals, plot = FALSE),
                  ylab="ACF", main="")
p2acf <- autoplot(acf(model.ar$residuals, plot = FALSE),
                  ylab="ACF", main="")
p1pacf <- autoplot(pacf(model.ma$residuals, plot = FALSE),
                  ylab="PACF", main="")
p2pacf <- autoplot(pacf(model.ar$residuals, plot = FALSE),
                  ylab="PACF", main="")

grid.arrange(p1, p2, p1acf, p2acf, p1pacf, p2pacf, nrow=3)
```



Neither of these models fit the data perfectly, and both use high order processes ($p = 6$ and $q = 5$ respectively), which include a relatively large number of parameters. This emphasises two important points:

1. Even if the correlation structure does not look like an AR(p) or an MA(q) process, fitting these models with large enough p and q will remove the majority of the correlation. Therefore it is better to model correlation with the wrong time series process than not to model it at all.
2. However, AR(p) or MA(q) processes are not always appropriate models for short-term correlation.

In this section we discuss a wider class of processes that encompass both AR(p) and MA(q) processes as special cases. The first type of process we describe is for modelling stationary data, while the second class can additionally model long-term trends.

Recall that an AR(p) process is given by

$$\begin{aligned} X_t &= \alpha_1 X_{t-1} + \ldots + \alpha_p X_{t-p} + Z_t \\ X_t - \alpha_1 X_{t-1} - \ldots - \alpha_p X_{t-p} &= Z_t \\ (1 - \alpha_1 B - \alpha_2 B^2 - \ldots - \alpha_p B^p) X_t &= Z_t \\ \phi(B) X_t &= Z_t \end{aligned}$$

while an MA(q) process is given by

$$\begin{aligned} X_t &= Z_t + \lambda_1 Z_{t-1} + \ldots + \lambda_q Z_{t-q} \\ &= (1 + \lambda_1 B + \ldots + \lambda_q B^q) Z_t \\ &= \theta(B) Z_t \end{aligned}$$

We generalise these time series models by combining them together.

Definition 1 (Autoregressive moving average process).

*An **autoregressive moving average process of order** (p, q) denoted ARMA(p, q) is given by*

$$\begin{aligned} X_t &= \alpha_1 X_{t-1} + \ldots + \alpha_p X_{t-p} + Z_t + \lambda_1 Z_{t-1} + \ldots + \lambda_q Z_{t-q} \\ &= \sum_{j=1}^{p} \alpha_j X_{t-j} + \sum_{j=1}^{q} \lambda_j Z_{t-j} + Z_t \end{aligned}$$

and is a combination of an AR(p) process and an MA(q) process. Using the backshift operator the model can be re-written as

$$\begin{aligned} X_t &= \sum_{j=1}^{p} \alpha_j X_{t-j} + \sum_{j=1}^{q} \lambda_j Z_{t-j} + Z_t \\ X_t - \sum_{j=1}^{p} \alpha_j X_{t-j} &= \sum_{j=1}^{q} \lambda_j Z_{t-j} + Z_t \\ (1 - \alpha_1 B - \ldots - \alpha_p B^p) X_t &= (1 + \lambda_1 B + \ldots + \lambda_q B^q) Z_t \\ \phi(B) X_t &= \theta(B) Z_t \end{aligned}$$

An ARMA(p, q) model is a more flexible representation of short-term correlation than using an AR(p) or an MA(q) process alone.

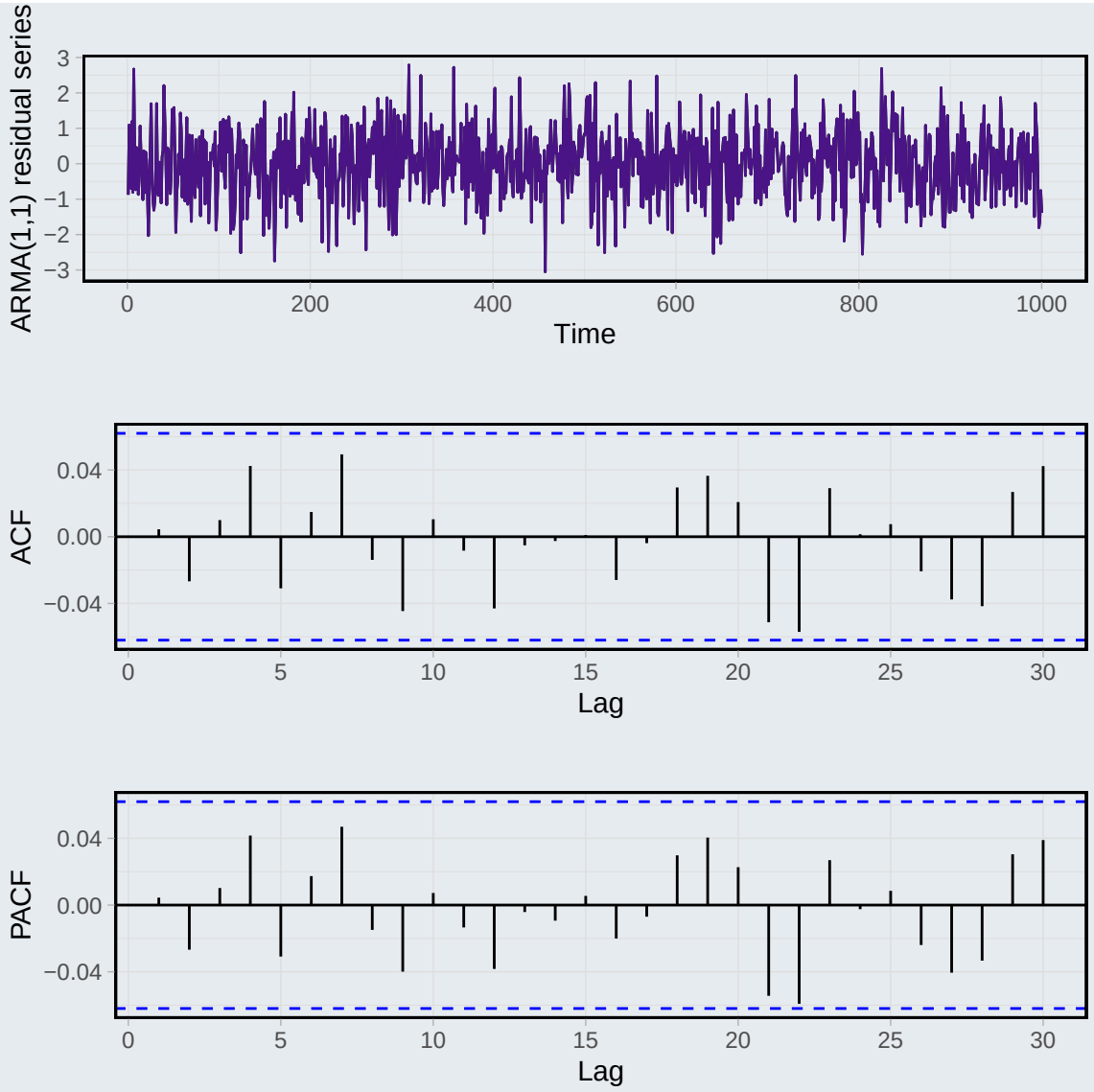
Example 2.

The data in this example can be modelled by an ARMA(1,1) process

$$X_t = \alpha X_{t-1} + \lambda Z_{t-1} + Z_t$$

using the R code:

```
model.arma <- arima(data, order=c(1,0,1))

# Plotting
p <- autoplot(model.arma$residuals, ylab="ARMA(1,1) residual series", main="") +
    geom_line( colour="#4a1486")
pacf <- autoplot(acf(model.arma$residuals, plot = FALSE),
                 ylab="ACF", main="")
ppacf <- autoplot(pacf(model.arma$residuals, plot = FALSE),
                  ylab="PACF", main="")
grid.arrange(p, pacf, ppacf, nrow=3)
```



Mean of an ARMA(p, q) process

The mean of an ARMA(p, q) process can be calculated in the same way as for an AR(p) process, using the following result.

Theorem 1.

Any ARMA(p, q) process can be written as an infinite sum of a purely random process, i.e. an MA(∞) process. That is

$$X_t = \sum_{j=1}^{p} \alpha_j X_{t-j} + \sum_{j=1}^{q} \lambda_j Z_{t-j} + Z_t = \sum_{j=0}^{\infty} \beta_j Z_{t-j}$$

for some coefficients β_j.

Supplementary material:
Variance and autocorrelation function

Calculating the variance and autocorrelation function for an ARMA(p, q) process can be done by assuming the process is stationary and deriving the Yule-Walker equations. However, this becomes algebraically messy for even small values of p and q. Therefore we illustrate the calculation for the ARMA(1,1) process

$$X_t = \alpha X_{t-1} + \lambda Z_{t-1} + Z_t$$

The variance is calculated as follows.

$$\begin{aligned} X_t &= \alpha X_{t-1} + \lambda Z_{t-1} + Z_t \\ X_t^2 &= \alpha X_{t-1} X_t + \lambda Z_{t-1} X_t + Z_t X_t \\ E(X_t^2) &= \alpha E(X_{t-1} X_t) + \lambda E(Z_{t-1} X_t) + E(Z_t X_t) \\ \mathrm{Var}(X_t) &= \alpha \gamma_1 + \lambda E(Z_{t-1} X_t) + E(Z_t X_t) \end{aligned}$$

where the last line holds true because $E(X_t) = 0$, so that $\mathrm{Var}(X_t) = E(X_t^2)$. This also means that $\gamma_1 = \mathrm{Cov}(X_{t-1} X_t) = E(X_{t-1} X_t)$. The latter two expectations are straightforward to calculate if we recall that

$$E(Z_t Z_{t-k}) = 0 \,\forall\, k > 0 \qquad \text{and} \qquad E(Z_t X_{t-k}) = 0 \,\forall\, k > 0$$

which are true because Z_t is a purely random process. Then

$$\begin{aligned} E(Z_t X_t) &= E(Z_t \{\alpha X_{t-1} + \lambda Z_{t-1} + Z_t\}) \\ &= E(Z_t^2) \\ &= \sigma_z^2 \end{aligned}$$

and

$$\begin{aligned} E(Z_{t-1} X_t) &= E(Z_{t-1} \{\alpha X_{t-1} + \lambda Z_{t-1} + Z_t\}) \\ &= \alpha E(Z_{t-1} X_{t-1}) + \lambda E(Z_{t-1}^2) \\ &= \alpha \sigma_z^2 + \lambda \sigma_z^2 \end{aligned}$$

Therefore we get that

$$\begin{aligned} \mathrm{Var}(X_t) &= \alpha \gamma_1 + \lambda(\alpha \sigma_z^2 + \lambda \sigma_z^2) + \sigma_z^2 \\ &= \alpha \gamma_1 + \sigma_z^2(\lambda^2 + \lambda \alpha + 1) \end{aligned}$$

The autocorrelation function at lag 1 is calculated in the same way,

$$\begin{aligned} X_t &= \alpha X_{t-1} + \lambda Z_{t-1} + Z_t \\ X_{t-1} X_t &= \alpha X_{t-1}^2 + \lambda X_{t-1} Z_{t-1} + X_{t-1} Z_t \\ E(X_{t-1} X_t) &= \alpha E(X_{t-1}^2) + \lambda E(X_{t-1} Z_{t-1}) + E(X_{t-1} Z_t) \\ \gamma_1 &= \alpha \gamma_0 + \lambda \sigma_z^2 \end{aligned}$$

Dividing by $\gamma_0 = \mathrm{Var}(X_t)$ gives the autocorrelation function at lag 1

$$\begin{aligned} \rho_1 &= \alpha + \frac{\lambda \sigma_z^2}{\gamma_0} \\ &= \alpha + \frac{\lambda \sigma_z^2}{\alpha \gamma_1 + \sigma_z^2(\lambda^2 + \lambda \alpha + 1)} \end{aligned}$$

The lag τ autocorrelation function for any $\tau > 1$ is calculated analogously as

$$
\begin{aligned}
X_t &= \alpha X_{t-1} + \lambda Z_{t-1} + Z_t \\
X_{t-\tau} X_t &= \alpha X_{t-\tau} X_{t-1} + \lambda X_{t-\tau} Z_{t-1} + X_{t-\tau} Z_t \\
E(X_{t-\tau} X_t) &= \alpha E(X_{t-\tau} X_{t-1}) + \lambda E(X_{t-\tau} Z_{t-1}) + E(X_{t-\tau} Z_t) \\
\gamma_\tau &= \alpha \gamma_{\tau-1}
\end{aligned}
$$

Therefore the autocorrelation function is given by

$$\rho_\tau = \alpha \rho_{\tau-1}$$

which is the same Yule-Walker equation as an AR(1) process. Therefore we have

$$
\begin{aligned}
\rho_0 &= 1 \\
\rho_1 &= \alpha + \frac{\lambda \sigma_z^2}{\alpha \gamma_1 + \sigma_z^2 (\lambda^2 + \lambda \alpha + 1)} \\
\rho_2 &= \alpha^2 + \frac{\alpha \lambda \sigma_z^2}{\alpha \gamma_1 + \sigma_z^2 (\lambda^2 + \lambda \alpha + 1)}
\end{aligned}
$$

and so on. The autocorrelation function for a higher order ARMA(p, q) process can be calculated using the same method, although it becomes increasingly messy.

Stationarity and invertibility

We know that an MA(q) process is always stationary and an AR(p) process is always invertible. Therefore an ARMA(p, q) process

$$\phi(B) X_t = \theta(B) Z_t$$

- is stationary if the AR(p) part is stationary, i.e. if the roots of the AR(p) characteristic polynomial $\phi(B)$ have modulus larger than one;
- is invertible if the MA(q) part is invertible, i.e. if the roots of the MA(q) characteristic polynomial $\theta(B)$ have modulus larger than one.

Example 3.

Consider the ARMA(1,1) process

$$X_t = 2X_{t-1} - 0.4Z_{t-1} + Z_t$$

which can be re-written as

$$X_t(1 - 2B) = Z_t(1 - 0.4B)$$

The characteristic equation for the AR part is $\phi(B) = 1 - 2B = 0$, which has a single root $B = 0.5 < 1$. Therefore the process is not stationary. The characteristic equation for the MA part is $\theta(B) = 1 - 0.4B = 0$, which has a single root $B = 2.5 > 1$. Therefore the process is invertible.

Task 1.

For the following ARMA(p, q) models write down the characteristic equations and determine whether each process is stationary/invertible.

- $X_t = 5X_{t-1} + Z_t - 0.2Z_{t-1}$
- $X_t = X_{t-1} + Z_t - X_{t-2} + 0.1Z_{t-1}$

Model identification

AR(p) and MA(q) processes are straightforward to identify from the ACF and PACF functions as follows.

- If the ACF is significantly different from zero for only the first q lags (for small q), then an MA(q) model is appropriate.
- If the PACF is significantly different from zero for only the first p lags (for small p), then an AR(p) model is appropriate.

We can summarise the features or ACF and PACF plots for AR, MA and ARMA processes in the following table:

Process	ACF	PACF
AR(p)	Tails off as exponential decay or damped sine wave	Cuts off after lag p
MA(q)	Cuts off after lag q	Tails off as exponential decay or damped sine wave
ARMA(p, q)	Tails off after lag $(q - p)$	Tails off after lag $(p - q)$

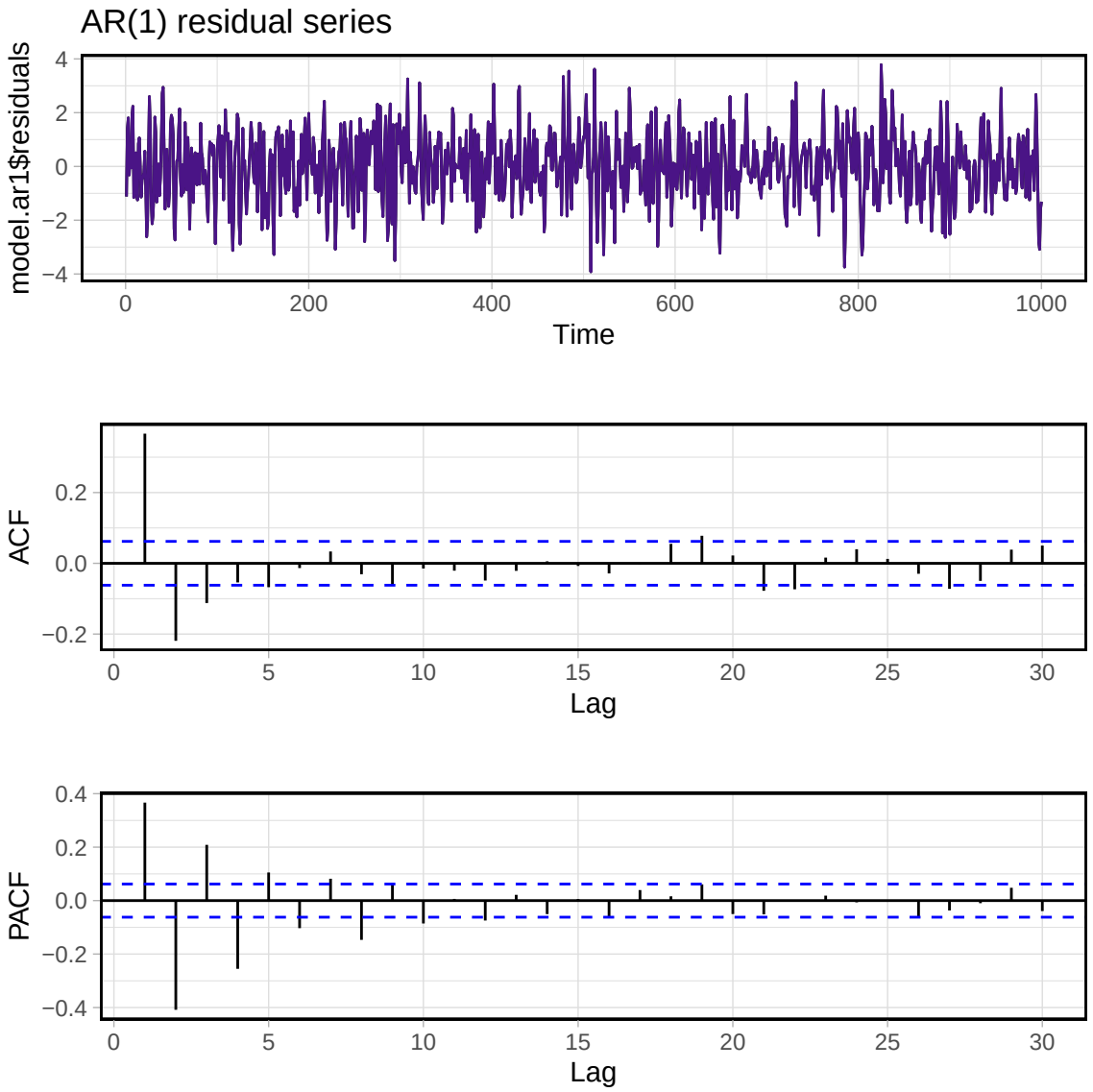
The pattern for ARMA means that identifying an ARMA model is not that easy. The data in the Example 1 are simulated from an ARMA(1,1) process. Notice that neither the ACF or PACF give any clues as to the appropriate type of time series process. All they tell us is that it is not an AR(p) process or an MA(q) process. Furthermore, we may hope that if we fit an AR(1) process to these data the residuals will look like an MA(1) process and if we fit an MA(1) process the residuals would resemble an AR(1) process, but as the graphs below show this is not the case.

```
# Fit an AR(1) model to the data
model.ar1 <- arima(data, order=c(1,0,0))
model.ar1
```

```

Call:
arima(x = data, order = c(1, 0, 0))

Coefficients:
         ar1  intercept
```

```
      0.8407     0.0025
s.e.  0.0172     0.2448

sigma^2 estimated as 1.535:  log likelihood = -1633.85,  aic = 3273.69
```

```r
# Plotting
p <- autoplot(model.ar1$residuals, main="AR(1) residual series") +
               geom_line(colour="#4a1486")
pacf <- autoplot(acf(model.ar1$residuals, plot = FALSE),
                 ylab="ACF", main="")
ppacf <- autoplot(pacf(model.ar1$residuals, plot = FALSE),
                  ylab="PACF", main="")
grid.arrange(p, pacf, ppacf, nrow=3)
```

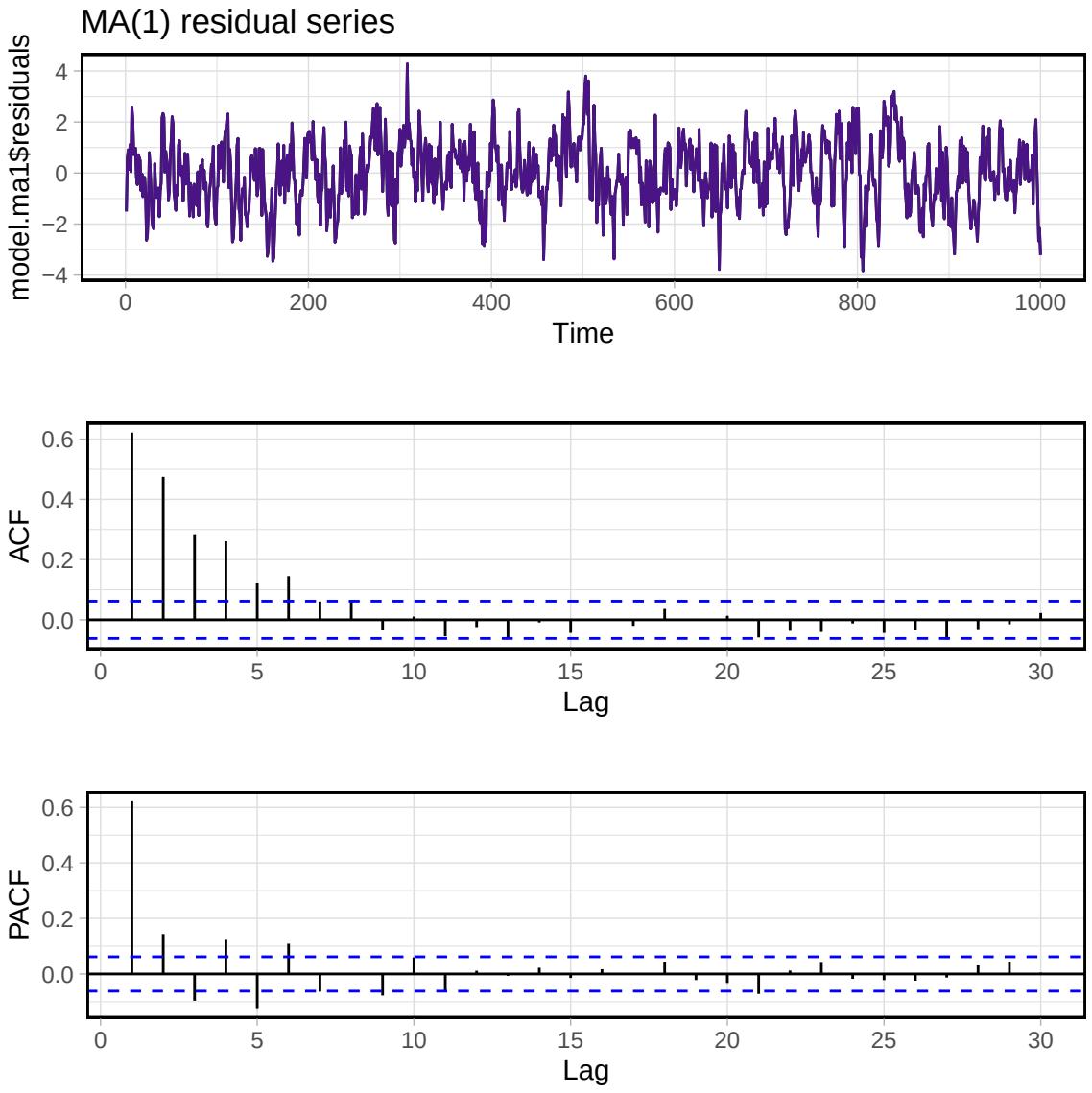



```r
# Fit an MA(1) model to the data
model.ma1 <- arima(data, order=c(0,0,1))
model.ma1
```

```
Call:
arima(x = data, order = c(0, 0, 1))

Coefficients:
         ma1  intercept
      0.975     0.0364
s.e.  0.009     0.0800
```

```
sigma^2 estimated as 1.641:  log likelihood = -1668.03,  aic = 3342.06
```

```
# Plotting
p <- autoplot(model.ma1$residuals, main="MA(1) residual series") +
               geom_line(colour="#4a1486")
acfp <- autoplot(acf(model.ma1$residuals, plot = FALSE),
                 ylab="ACF", main="")
pacfp <- autoplot(pacf(model.ma1$residuals, plot = FALSE),
                  ylab="PACF", main="")
grid.arrange(p, acfp, pacfp, nrow=3)
```



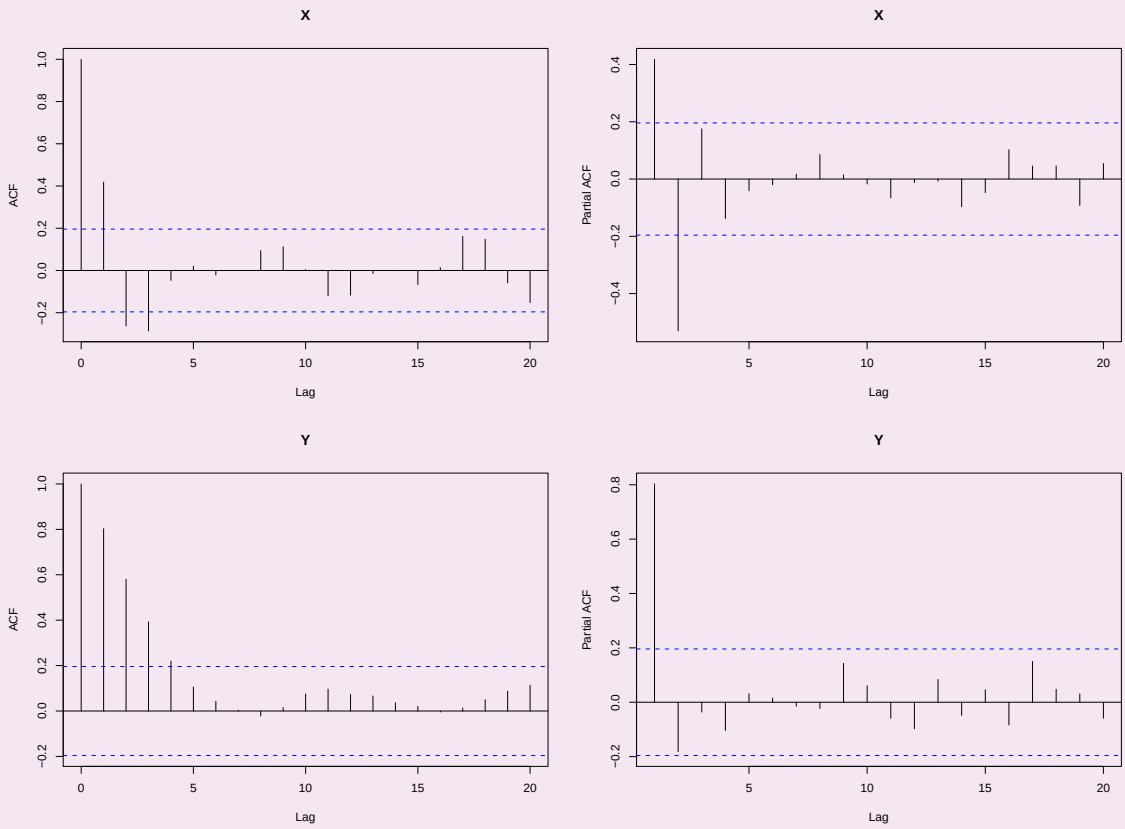
Notes

1. Model identification for an ARMA(p, q) process where $p, q > 0$ is difficult.
2. First determine if the ACF and PACF resemble either an MA(q) or an AR(p) process.
3. If not then adopt a trial and error approach, starting with the simplest model (i.e. an ARMA(1,1)) and increasing the complexity until the correlation has been removed.

Task 2.

How would you go about finding an appropriate ARMA(p, q) model for the time series processes X and Y

with ACF and PACF plots shown below?



Simulation example

Week 6 discussed methods of removing trends and seasonal variation from time series data, while Week 7 described how to model short-term correlation in a stationary time series. But is modelling correlation important or can we simply ignore it?

We look at the consequences of ignoring correlation using simulation. Consider the simple linear trend model

$$X_t = \beta_0 + \beta_1 t + e_t$$

where e_t could be independent errors or correlated. The aim of the analysis is to estimate the slope coefficient β_1 and produce a 95% confidence interval. Two natural questions we could ask are:

1. What effect does the correlation structure of e_t have on the estimate and confidence interval of β_1?
2. If e_t is correlated, can we allow for this correlation when we estimate β_1?

We answer these questions using simulation.

Generating data

We generate time series of length 1000 from the following two models,

$$\begin{array}{lrcl} \textbf{A} & X_t & = & 30 + 0.1t + Z_t \\ \textbf{B} & X_t & = & 30 + 0.1t + Y_t \end{array}$$

where the regression parameter $\beta = 0.1$.

- Z_t is a purely random process, meaning that model **A** is a linear trend with independent errors.

- Y_t is an AR(1) process with lag one autocorrelation coefficient equal to 0.9, meaning that model **B** is a linear trend with correlated errors.

We simulate 1000 sets of data from models **A** and **B**, using the following R code.

```
time <- 1:1000
corr.ar1 <- arima.sim(model=list(ar=c(0.9)), n=1000, sd=50)
corr.indep <- arima.sim(model=list(),  n=1000, sd=50)[1:1000]
data.ar1 <- corr.ar1 + 30 + 0.1*time
data.indep <- corr.indep + 30 + 0.1*time
```

Measuring model quality

From 1000 simulated datasets we get 1000 estimates of β, $\hat{\beta}_1, \ldots, \hat{\beta}_{1000}$. But how good are they? There are three standard metrics for measuring this.

1. **Bias** - On average (over all 1000 simulated datasets) how different is β from the estimates, which is calculated as

$$\text{Bias}(\beta) = E(\hat{\beta}) - \beta = \frac{1}{1000}\sum_{j=1}^{1000}\hat{\beta}_j - \beta$$

2. **Root mean square error (RMSE)** - How much variation is there between the 1000 estimates, which is calculated as

$$\text{RMSE}(\beta) = \sqrt{E[(\hat{\beta}-\beta)^2]} = \sqrt{\frac{1}{1000}\sum_{j=1}^{1000}(\hat{\beta}_j-\beta)^2}$$

3. **Coverage probability** - Each dataset produces an estimate and 95% confidence interval for β. What percentage of the 95% confidence intervals contain the true value β?

Conducting the simulation study

We answer the two questions listed above as follows.

1. What effect does the correlation structure of e_t have on the estimate and confidence interval of β_1?

To answer this we naively assume that both datasets are independent, and estimate β_1 and its 95% confidence interval using the `lm()` function. Then we save the estimates and 95% confidence intervals from each data set and calculate the bias, RMSE and coverage probability.

First we start with fitting a linear trend model to independent data:

```
# Specify the length of the time series
time <- 1:1000
n <- length(time)

# Specify the linear trend and error variance
beta0 <- 30
beta1 <- 1
Z.sd <- 50

# Create a matrix in which to save the estimate
# and whether or not the CI contains the true value
n.simulation <- 1000
results <- array(NA, c(1000,2))

# Run the simulation
for(k in 1:n.simulation)
{
# Generate the simulated data
X <- beta0 + beta1 * time + rnorm(n=n, mean=0, sd=Z.sd)
```

```
#plot(time, X)

# Estimate beta1 by least squares
model <- lm(X~time)
#summary(model)

# Save the estimate and CI for beta1
results[k,1] <- model$coefficients[2]
SE <- sqrt(summary(model)$cov.unscaled[2,2]) * summary(model)$sigm
CI <- c(results[k,1] - 1.96*SE, results[k,1] + 1.96*SE)
results[k, 2] <- as.numeric(beta1>CI[1] & beta1 < CI[2])
}
```

Next we calculate the bias, MSE and coverage:

```
# Bias
mean(results[ ,1])-beta1
```

```
[1] -0.0002430945
```

```
# RMSE
sqrt(mean((results[ ,1]-beta1)^2))
```

```
[1] 0.005316891
```

```
# Coverage
100*sum(results[ ,2])/n.simulation
```

```
[1] 95.4
```

The bias and RMSE seem small and the coverage probability is close to 95%.

Next we fit a linear trend model to AR(1) data by following the same steps as before, with the only difference being the errors which are now simulated from a different process.

```
results <- array(NA, c(1000,2))

# Run the simulation
for(k in 1:n.simulation)
{
# Generate the simulated data
X <- beta0 + beta1 * time + as.numeric(arima.sim(model=list(ar=c(0.9)), n=n, sd=Z.sd))

# Estimate beta1 by least squares
model <- lm(X~time)

# Save the estimate and CI for beta1
results[k,1] <- model$coefficients[2]
SE <- sqrt(summary(model)$cov.unscaled[2,2]) * summary(model)$sigm
CI <- c(results[k,1] - 1.96*SE, results[k,1] + 1.96*SE)
results[k, 2] <- as.numeric(beta1>CI[1] & beta1 < CI[2])
}

# Calculate the bias, MSE and coverage
# Bias
mean(results[ ,1]) - beta1
```

```
[1] 0.0004257257
```

```
# RMSE
sqrt(mean((results[ ,1]-beta1)^2))
```

```
[1] 0.05477506
```

```
# Coverage
100*sum(results[ ,2]) / n.simulation
```

```
[1] 33.1
```

Notice that although the bias and RMSE are not too different from the values we got for the previous model, the coverage probability is much lower than 95%.

2. If e_t is correlated, can we allow for this correlation when we estimate β_1?

To answer this we can simultaneously estimate the linear trend and model the correlation using the `arima()` function. The R code to do this is as follows.

```
model.ar1 <- lm(data.ar1~time)
arima(data.ar1, order=c(1,0,0), xreg=time)
```

```
Call:
arima(x = data.ar1, order = c(1, 0, 0), xreg = time)

Coefficients:
         ar1  intercept    time
      0.8875    22.6929  0.1246
s.e.  0.0145    26.3488  0.0454

sigma^2 estimated as 2279:  log likelihood = -5285.5,  aic = 10579
```

where the `arima()` function has an `xreg` argument which can be given a vector or matrix of regression variables, which in this case is a linear function of time. Again we save the estimates and 95% confidence intervals from each data set and calculate the bias, RMSE and coverage probability.

```
# Fit a linear trend model to AR(1) data and allow for AR(1) correlation

results <- array(NA, c(1000,2))

# Run the simulation
for(k in 1:n.simulation)
{
# Generate the simulated data
X <- beta0 + beta1 * time + as.numeric(arima.sim(model=list(ar=c(0.9)), n=n, sd=Z.sd))

# Estimate beta1 by least squares
arima.ar1 <- arima(X, order=c(1,0,0), xreg=time)

# Save the estimate and CI for beta1
results[k,1] <- arima.ar1$coef[3]

SE <- sqrt(arima.ar1$var.coef[3,3])
CI <- c(results[k,1] - 1.96*SE, results[k,1] + 1.96*SE)
results[k, 2] <- as.numeric(beta1>CI[1] & beta1 < CI[2])
}

# Calculate the bias, MSE and coverage
# Bias
mean(results[ ,1]) - beta1
```

```
[1] 0.001110324
```

```
# MSE
sqrt(mean((results[ ,1]-beta1)^2))
```

```
[1] 0.05228679
```

```
# Coverage
100*sum(results[ ,2]) / n.simulation
```

```
[1] 93.5
```

The bias and RMSE are similar to the previous values, but the coverage is now much closer to 95%.

Therefore if we want to model trend and correlation in time series data using regression methods, the following strategy is appropriate.

1. First remove any trend and seasonal variation assuming the observations are independent using the `lm()` function, because at this stage we do not know if there is any correlation.
2. Determine whether the residuals have any short-term correlation, and if so what type of stationary time series model is appropriate.
3. Finally, simultaneously estimate the correlation and trend using the `arima()` function.

Non-stationary models

So far, we have modelled trend and seasonal variation first, before representing the residuals with a short-term correlation model. This is the approach most commonly adopted in time series modelling, because it allows the shape of the trend and seasonal variation to be estimated, before representing the correlation structure with a fairly simple time series model.

An alternative approach is to model the trend, seasonal variation and correlation simultaneously. However, although such an approach has been widely used, it simply removes the trend rather than modelling it. Therefore, if capturing the shape of the trend or seasonal variation is the goal of the analysis, this approach is not appropriate.

The class of non-stationary time series models described here combine ARMA(p, q) models and differencing. Recall from Week 6 that one way to eliminate a trend in a non-stationary time series X_t, is to calculate its first order difference

$$Y_t = \nabla X_t = (1 - B)X_t = X_t - X_{t-1}.$$

Usually, first or second order differences are enough to obtain a stationary series $\{Y_t\}$, but in general we can difference d times:

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

Combining this operator with an ARMA(p, q) process leads to the following general class of models.

Definition 2 (Autoregressive integrated moving average process).

*$\{X_t\}$ is an **autoregressive integrated moving average process of order (p, d, q)**, denoted ARIMA(p, d, q) if the dth order differenced process*

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

is an ARMA(p, q) process. An ARIMA(p, d, q) process can be written most easily in terms of characteristic polynomials. If we write the ARMA(p, q) process for Y_t as

$$\phi(B)Y_t = \theta(B)Z_t$$

then as $Y_t = (1 - B)^d X_t$, an ARIMA(p, d, q) process can be written as

$$\phi(B)(1 - B)^d X_t = \theta(B)Z_t.$$

Notes

- The characteristic polynomial for the AR part of the ARIMA model is equal to $\phi^*(B) = \phi(B)(1 - B)^d$, which has d roots that equal 1. Hence an ARIMA(p, d, q) process cannot be stationary unless $d = 0$.
- The parameter d controls the number of times the process is differenced, and must be a non-negative integer.
- When $d = 0$ we have an ARMA(p, q) model, i.e. ARIMA(p, 0, q) = ARMA(p, q).

Example 4.

For an ARIMA(1,1,1) process the characteristic polynomials are given by $\phi(B) = 1 - \alpha B$ and $\theta(B) = 1 + \lambda B$, meaning that the full model is given by

$$\begin{aligned}(1-\alpha B)(1-B)X_t &= (1+\lambda B)Z_t \\ (1-B-\alpha B+\alpha B^2)X_t &= \lambda Z_{t-1} + Z_t \\ X_t &= (1+\alpha)X_{t-1} - \alpha X_{t-2} + \lambda Z_{t-1} + Z_t\end{aligned}$$

So an ARIMA(1,1,1) model is essentially a non-stationary ARMA(2,1) model.

Task 3.

Consider the ARIMA time series process

$$(1-B)(1-0.2B)X_t = (1-0.5B)Z_t$$

- What type of process is defined here (i.e. what are p, d, q)?
- Write out the model in terms of X_t.
- Is this process stationary?

Task 4.

Consider an ARIMA$(0, 2, 1)$ process.

- Write out the model in backshift (B) notation.
- Expand this equation and write the model in terms of X_t.

Task 5.

Consider the following ARIMA processes.

- ARIMA(1,0,0)
- ARIMA(0,0,2)
- ARIMA(0,1,0)

Which of these are stationary?

Model identification

To model trend and correlation in a single model using a non-stationary ARIMA(p, d, q) process the following four step approach seems reasonable.

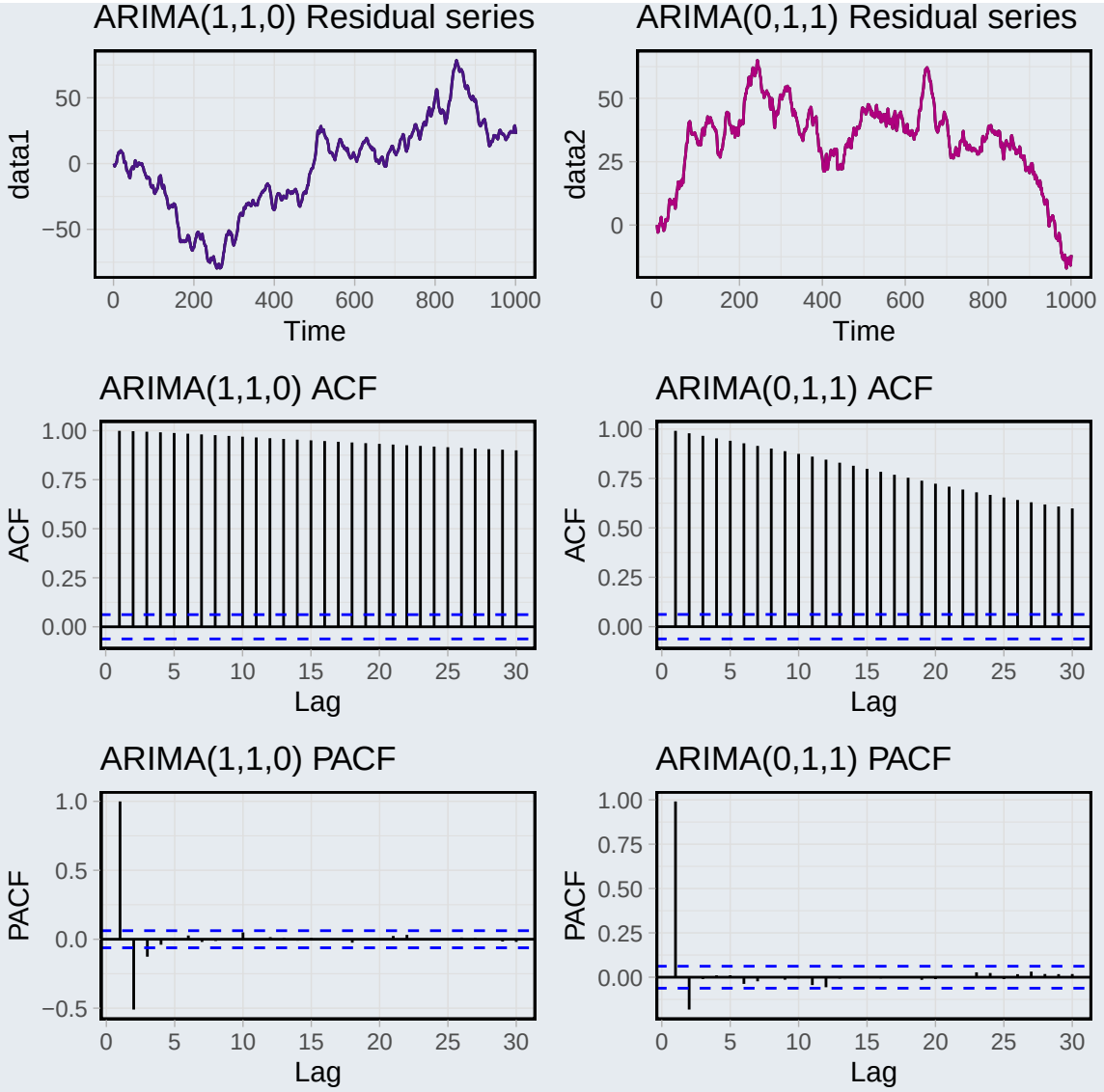
1. **Choose** d: Plot the time series and its correlogram, and determine whether the data contain a trend and hence need to be differenced. If there is no trend then choose $d = 0$, otherwise difference the data and plot the differenced data. If this looks stationary then choose $d = 1$, otherwise difference again and plot the second differences. Repeat this process until the data are stationary. Typically $d = 1$ or $d = 2$ should be enough to obtain a stationary time series.
2. **Choose** p **and** q: Plot the ACF and PACF of the dth order differences, and determine the appropriate ARMA(p, q) model.
3. **Estimate the parameters**: Use the `arima()` function in R to estimate the parameters of the ARIMA process.
4. **Residual diagnosis**: Look at the time plot, ACF and PACF plot of the residuals and determine whether they contain any remaining trend, seasonal variation or short-term correlation. If the residuals resemble a purely random

process then stop, otherwise return to stage one and change either p, d or q.

Example 5.

The following plot shows realisations of an ARIMA(1,1,0) (left column) and ARIMA(0,1,1) (right column) processes. Note the data are non-stationary and have a trend. Therefore the ACF is not informative regarding the presence or absence of short-term correlation.

```
# Simulate ARIMA(1,1,0) and ARIMA(0,1,1) data
data1 <- arima.sim(model=list(ar=c(0.7), order=c(1,1,0)), n=1000, sd=1)
data2 <- arima.sim(model=list(ma=c(0.7), order=c(0,1,1)), n=1000, sd=1)

# Plotting
p1 <- autoplot(data1, main="ARIMA(1,1,0) Residual series") +
      geom_line(colour="#4a1486")

p2 <- autoplot(data2, main="ARIMA(0,1,1) Residual series") +
      geom_line(colour="#ae017e")

p1acf <- autoplot(acf(data1, plot = FALSE), main="ARIMA(1,1,0) ACF")
p2acf <- autoplot(acf(data2, plot = FALSE), main="ARIMA(0,1,1) ACF")

p1pacf <- autoplot(pacf(data1, plot = FALSE), main="ARIMA(1,1,0) PACF")
p2pacf <- autoplot(pacf(data2, plot = FALSE), main="ARIMA(0,1,1) PACF")

grid.arrange( p1, p2, p1acf, p2acf, p1pacf, p2pacf, nrow=3)
```



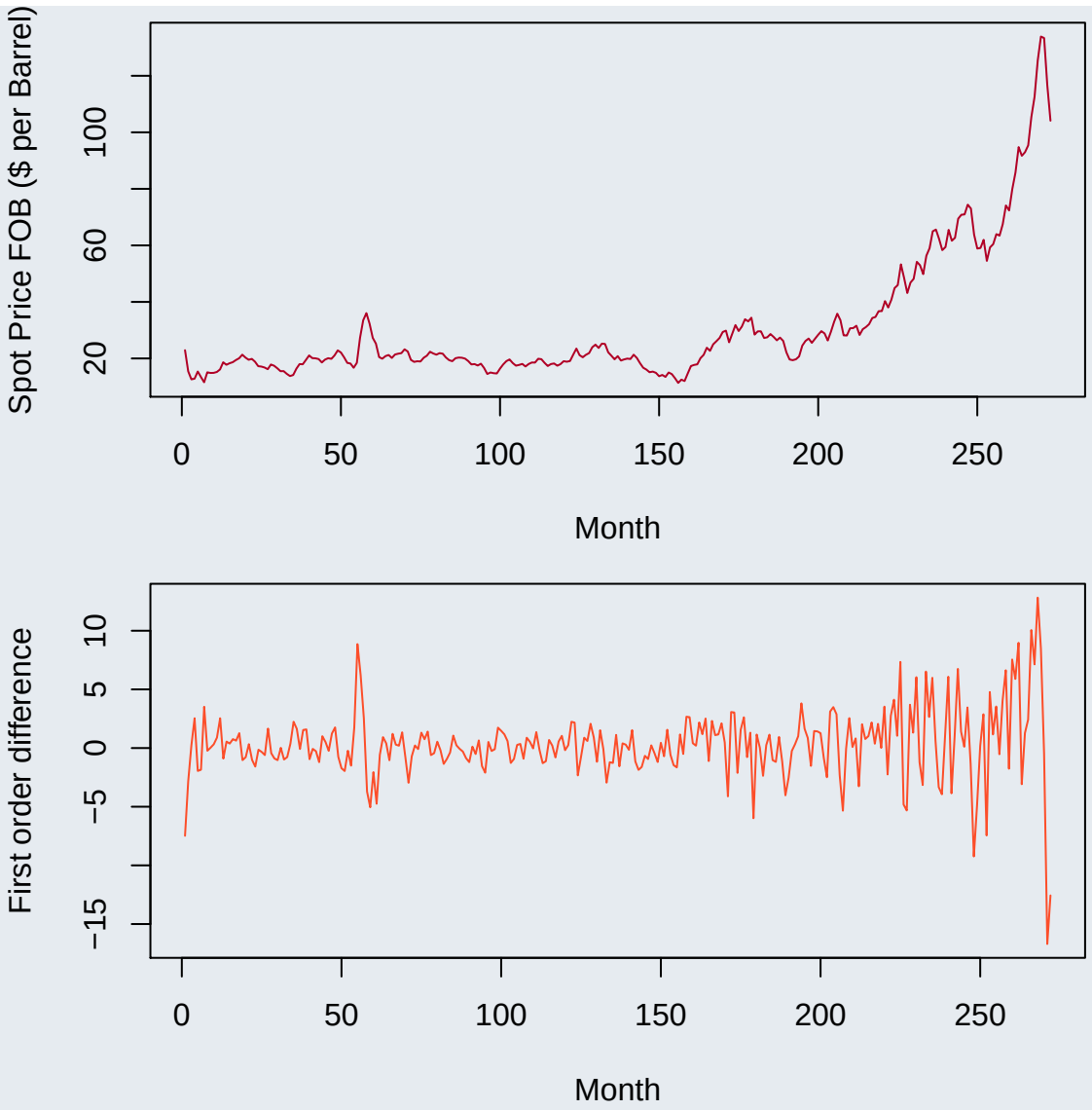
Example 6.

In this example, we look at a time series of monthly crude oil prices from 1986-2008 and we want to see if we can model the trend and correlation using an ARIMA(p, d, q) model.

```
crude <- read.csv(url("http://www.stats.gla.ac.uk/~tereza/rp/crude_oil.csv"))
price <- crude$spot_price_fob
```

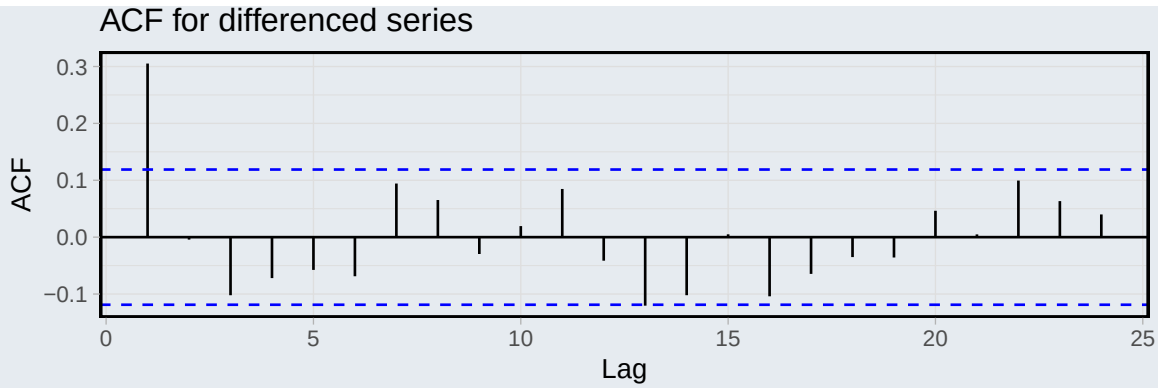
The following graphs show the original data which is clearly not stationary. Taking the first order difference seems to make the series stationary with zero mean. Therefore $d = 1$ appears to be adequate.

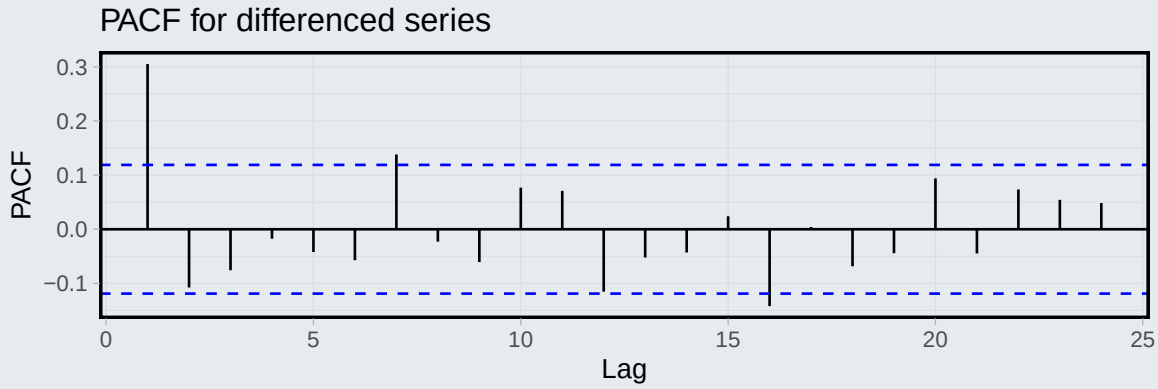
```
diff1 <-diff(price, differences = 1)
par( mfrow=c(2,1), mar=c(4,4,1,1))
plot(price, type="l", xlab = "Month", ylab = "Spot Price FOB ($ per Barrel)",
      col="#b10026")
plot(diff1, type="l", xlab = "Month", ylab = "First order difference",
      col="#fc4e2a")
```



The ACF and PACF of the first order differences are given below, and show that there is still some short term correlation left: both the ACF and PACF show siginificant correlation at lag 1.

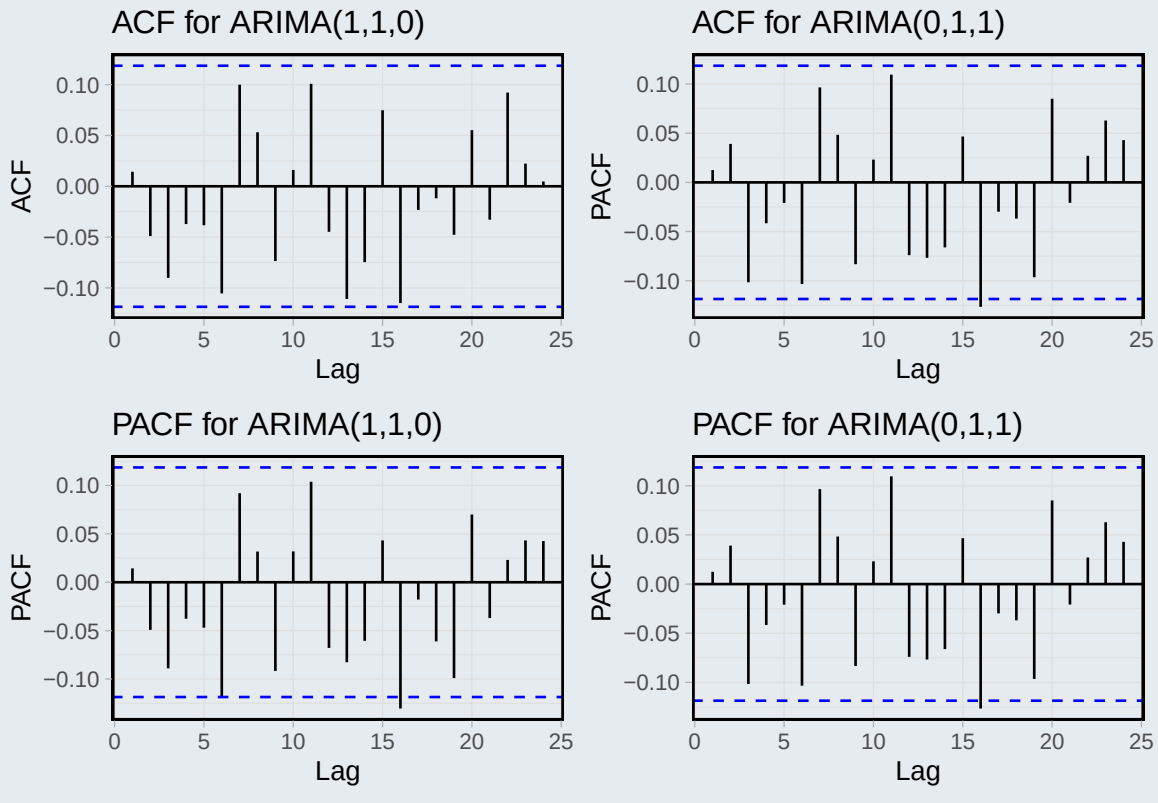
```
p1<- autoplot(acf(diff1, plot=FALSE), main="ACF for differenced series")
p2<- autoplot(pacf(diff1, plot=FALSE), main="PACF for differenced series")
grid.arrange(p1,p2, nrow=2)
```





We could now fit either an AR(1) or an MA(1) to this data to remove the short term correlation remaining after differencing the series, so we could fit either an ARIMA(1,1,0) or an ARIMA(0,1,1) to the original data.

```
arima110 <- arima(price, order = c(1,1,0))
arima011 <- arima(price, order = c(0,1,1))
p1<- autoplot(acf(arima110$residuals, plot=FALSE), main="ACF for ARIMA(1,1,0)")
p2<- autoplot(pacf(arima110$residuals, plot=FALSE), main="PACF for ARIMA(1,1,0)")
p3<- autoplot(pacf(arima011$residuals, plot=FALSE), main="ACF for ARIMA(0,1,1)")
p4<- autoplot(pacf(arima011$residuals, plot=FALSE), main="PACF for ARIMA(0,1,1)")
grid.arrange(p1,p3,p2,p4, nrow=2)
```



The residuals from both models appear to resemble a purely random process with no correlation, so both models seem to be appropriate. In general, if there are multiple candidate models to choose from, one could use model selection criteria to choose the best one. In this scenario, if we look at the AIC, the ARIMA(1,1,0) model seems to be marginally better than the ARIMA(0,1,1) model.

```
arima110$aic
```

```
[1] 1345.689
```

```
arima011$aic
```

```
[1] 1345.944
```

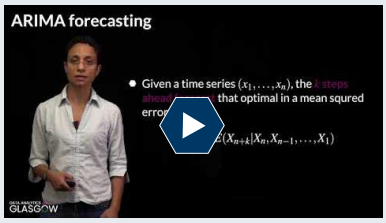
It is also possible to use the `auto.arima()` function from `library(forecast)` which will run a search and return the model with the smallest AIC (or alternative criterion).

Task 6.

Find a suitable ARIMA model for the `Nile` data, which gives measurements of the annual flow of the river Nile at Aswan in the period 1871–1970. For more information on the dataset type `?Nile` in R.

Forecasting

Our final topic in this introduction to time series analysis is forecasting.



Time series forecasting

https://youtu.be/Gr9uRCK9fSI

Duration: 08m02s

Forecasting is also called prediction, and involves predicting the value of a time series at future points in time. This makes it a very hard task, and all predictions should be accompanied with a measure of uncertainty. The majority of forecasting methods are based on a statistical model, so if the model is not appropriate, then the forecasts will be useless. Even if an appropriate model is fitted, it does not mean the forecasts will be reasonable. However, despite these reservations forecasting is vitally important, and is often the sole goal of a time series analysis. A few examples of problems which require forecasting are given below.

- Decisions about hiring / firing staff at a company will depend on predictions of future profits.
- Supermarkets need to predict food sales for the next week so they can decide how much to order.
- Environmental scientists are currently trying to forecast global temperature for the next 100 years to predict the effects of global warming.
- City traders want to predict the value of stocks and shares tomorrow so they can decide whether to buy or sell to make a profit.
- Governments need to predict the changing shifts in population demographics (e.g. how many old /young people) so that adequate provision for schools and nursing homes can be made.

A number of methods for predicting future observations are available, and we describe three of them here. But first a word of warning. Forecasting involves predicting the future and is hence very difficult. Do not simply believe that any forecast you see will be accurate.

General problem

A time series has been observed at n time points $(x_1, \ldots, x_n)$, and predictions are required for the series at times $n+1$, $n+2$, etc.

We denote the **k-step ahead forecast** of x_{n+k} given data $(x_1, \ldots, x_n)$ by $x_n(k)$, so that $x_n(1)$ is the prediction of x_{n+1} based on data up to and including time n.

The **forecast error** is given by

$$e_n(k) = x_{n+k} - x_n(k)$$

and is the amount by which the forecast differs from the true observation (once it has become available). The amount of uncertainty in a forecast is measured by the size of its error variance, $\mathrm{Var}(e_n(k))$, with larger values meaning the forecast is less reliable.

To evaluate the performance of a forecasting method on a given data set, we calculate 1 step ahead forecasts $x_1(1), \ldots, x_{n-1}(1)$, and measure the discrepancy to the observed values $x_2, \ldots, x_n$ using the **root mean square prediction error**

$$\text{RMSPE} = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n-1} e_k(1)^2} = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n-1}(x_{k+1} - x_k(1))^2}$$

Here we compare three methods of forecasting, regression, exponential smoothing and ARIMA models.

Regression

One approach is to ignore the temporal correlation in the observed data, and predict the next value of the time series based on linear regression methods. This method will only produce good forecasts if the time series being predicted has a strong trend and seasonal component compared to the amount of random variation and short-term correlation. For the additive time series model

$$X_t = m_t + s_t + e_t$$

the trend and seasonal variation are represented by

$$m_t + s_t = \mathbf{z}_t^\top \boldsymbol{\beta}$$

a linear regression model as described in chapter 2. The 1 step ahead prediction is then given by

$$x_n(1) = \mathbf{z}_{n+1}^\top \hat{\beta}$$

where $\hat{\beta}$ is the vector of regression parameter estimates and $\mathbf{z}_{n+1}^\top$ are the covariate values at time $n+1$. Approximate 95% prediction intervals can be calculated from linear model theory as

$$\mathbf{z}_{n+1}^\top \hat{\beta} \pm 1.96\sqrt{\hat{\sigma}^2(1 + \mathbf{z}_{n+1}^\top (Z^\intercal Z)^{-1}\mathbf{z}_{n+1})}$$

where $\hat{\sigma}^2$ is the estimated residual variance from the linear model, and Z is the matrix of regression variables for all n time points.

The above prediction interval is constructed under the assumption that the forecast errors are normally distributed. In cases where this assumption is unreasonable, one alternative is to use bootstrapping, which only assumes that the forecast errors are uncorrelated.

A forecast error is defined as $e_t = x_t - \hat{x}_{t|t-1}$ which can be re-written as:

$$x_t = \hat{x}_{t|t-1} + e_t$$

We can thus simulate the next observation of a time series using:

$$x_{n+1} = x_n(1) + e_{n+1}$$

where $x_n(1)$ is the one-step forecast and e_{n+1} is the unknown future error. Assuming that future errors will be similar to past errors, we can replace e_{n+1} by sampling with replacement from the collection of errors we have seen in the past (i.e. the residuals). Adding the new simulated observation to the time series, we can repeat the process to obtain:

$$x_{n+2} = x_n(2) + e_{n+2}$$

where e_{n+2} is another draw from the collection of residuals. Continuing this way, we can simulate the entire set of future values for the time series.

By repeating the above process many times, we obtain many possible sequences of future values. We can then compute the prediction intervals by calculating the percentiles for each forecast horizon. This is known as the bootstrapped prediction interval. This bootstrapping approach is only valid if the forecast errors are uncorrelated. In cases where this assumption is not plausible, block bootstrap can be employed, in which we replicate the correlation in the errors by resampling blocks of data instead.

Example 7.

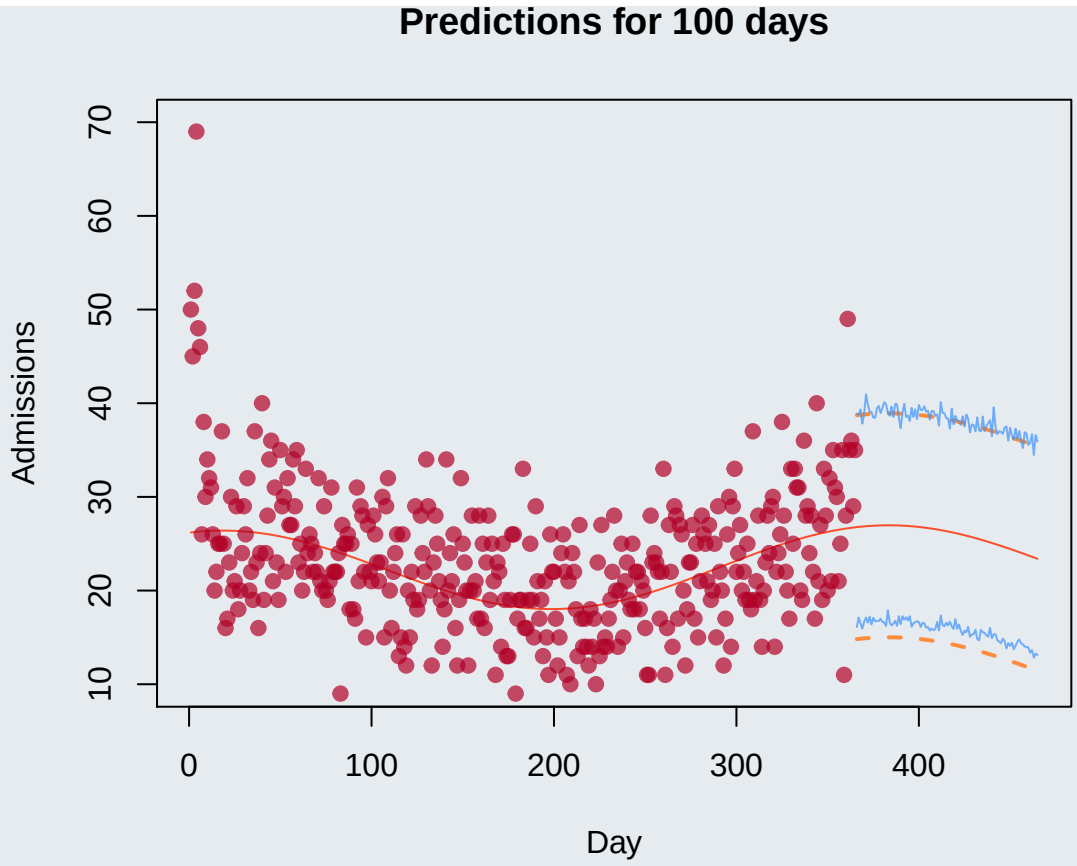
Consider the daily respiratory admissions data presented in Week 6. One of the trend models for these data was

$$X_t = \beta_0 + \beta_1 t + \beta_2 \sin(2\pi t/365) + \beta_3 \cos(2\pi t/365) + e_t$$

which modelled the regular seasonal pattern with a period of a year and a linear trend. The graph below shows the last year of these data, together with predictions for the first 100 days in 2008 using the model above. Note in this case how wide the prediction intervals are, because the seasonal pattern is overwhelmed by random variation. Thus the predictions are not likely to be very accurate. Also note the very small differences between the prediction interval based on the normality assumption of the errors (red) and the bootstrap prediction interval (blue). This is because the distribution of the residuals seem not to deviate much from normality.

The R code used to fit the model is given below.

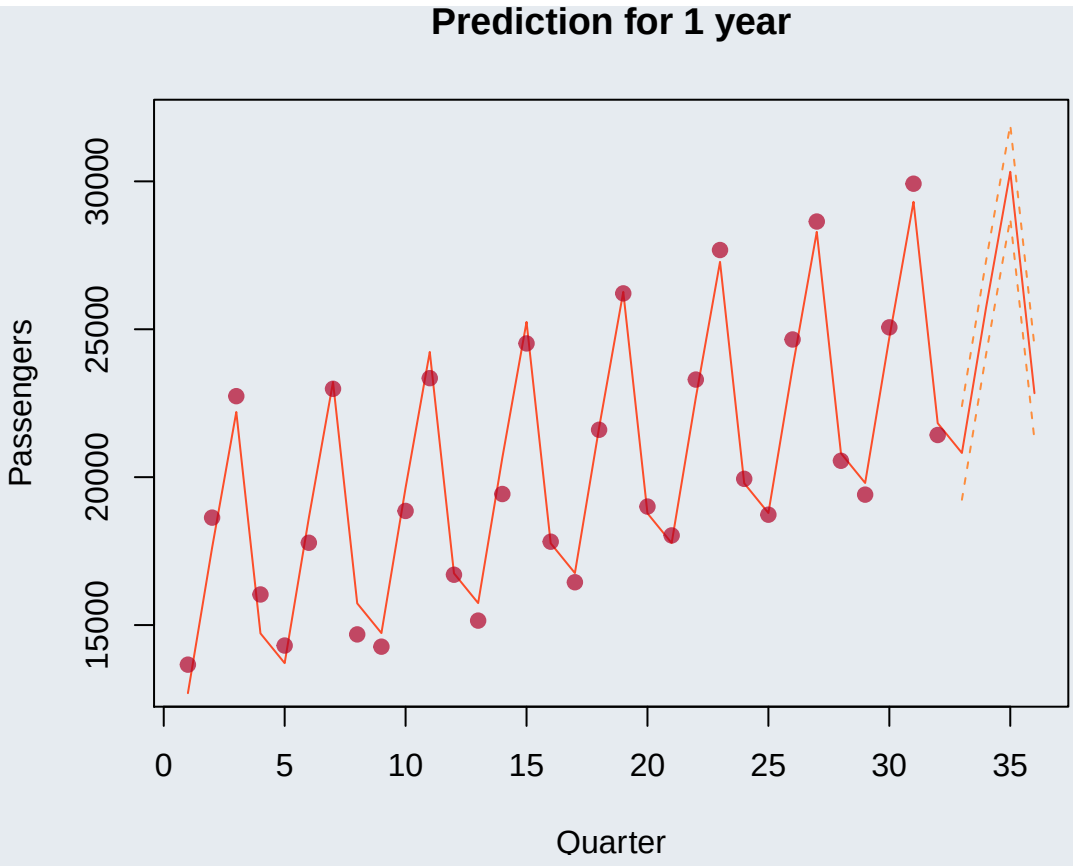
```
# Fit a sinusoidal model
data <- read.csv(url("http://www.stats.gla.ac.uk/~tereza/rp/resp.csv"))
x <- data[ ,2]
n <- length(x)
t <- 1:n
Z <- cbind(t, sin(2*pi*t/365), cos(2*pi*t/365))
z1 <- Z[ ,1]
z2 <- Z[ ,2]
z3 <- Z[ ,3]
model <- lm(x~z1+z2+z3)

# Predict the next 100 time points
t.predict <- (n+1):(n+100)
Z.predict <- data.frame(t.predict, sin(2*pi*t.predict/365), cos(2*pi*t.predict/365))
model.predict <- predict(model, newdata=data.frame(z1 = Z.predict[ , 1],
z2=Z.predict[ ,2], z3=Z.predict[ ,3]), se.fit = TRUE, interval="prediction")

# Bootstrap Prediction intervals
nboot <- 1000
bootfit <- matrix(NA,nrow=100,ncol=1000)
for (i in 1:nboot){
  bootfit[,i] <- sample(residuals(model),size=100)
  bootfit[,i] <- predict(model,
                         newdata=data.frame(z1 = Z.predict[ , 1],z2=Z.predict[ ,2],
                                            z3=Z.predict[,3]),se.fit=FALSE)+bootfit[,i]
}
PI.lwr <- apply(bootfit,1, function(x) quantile(sort(x),0.025))
PI.upr <- apply(bootfit,1, function(x) quantile(sort(x),0.975))


# Plot the predictions
plot(1:465,c(model$fitted.values[2558:n], model.predict$fit[ ,1]), type="l",
     xlab="Day", ylab="Admissions", ylim=c(10,70),
     main="Predictions for 100 days", col="#fc4e2a")
points(1:365, data[2558:n ,2], pch=19, col=alpha("#b10026",0.7))
lines(366:465, model.predict$fit[ ,2], lty=2, col="#fd8d3c",lwd=2)
lines(366:465, model.predict$fit[ ,3], lty=2, col="#fd8d3c",lwd=2)

lines(366:465, PI.lwr, lty=1, col=alpha("#3c93fd",0.7))
lines(366:465, PI.upr, lty=1, col=alpha("#3c93fd",0.7))
```



Example 8.

Consider the air traffic data, which can be modelled using a seasonal indicator variable for quarter and a linear trend. The graph below shows the data together with predictions for the next four quarters. Note that in this case the data are dominated by trend and seasonal variation, and have relatively little unexplained variation. Therefore the prediction intervals are narrow, suggesting the predictions will be fairly accurate.

```
library(zoo)
airtraffic<-read.csv(url("http://www.stats.gla.ac.uk/~tereza/rp/airtraffic.csv"))
# Create a quarterly date variable
airtraffic$Date <- as.yearqtr(paste(airtraffic$Year, airtraffic$Quarter),
                              format="%Y %q")

x <- airtraffic[ ,3]
quarter <- rep(1:4,8)
time <- 1:32
model <- lm(x~time+as.factor(quarter))

time.predict<- 33:36
model.predict<- predict(model, newdata=data.frame(time<- time.predict, quarter=1:4),
                        se.fit = TRUE, interval="prediction")

plot(c(1:36), c(model$fitted.values, model.predict$fit[ ,1]), type="l",
     ylim=c(13000, 32000), ylab="Passengers", main="Prediction for 1 year",
     xlab="Quarter", col="#fc4e2a")
points(x, pch=19, col=alpha("#b10026",0.7))
lines(33:36, model.predict$fit[ ,2], col="#fd8d3c", lty=2)
lines(33:36, model.predict$fit[ ,3], col="#fd8d3c", lty=2)
```



Example 9.

Consider the following linear trend model

$$X_t = 10 + 2t + \epsilon_t \qquad t = 1, \ldots, 10$$

where $\hat{\beta} = (10, 2)$ and $\hat{\sigma}^2 = 1$. The design matrix is given by

$$Z = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ \vdots & \vdots \\ 1 & 10 \end{pmatrix}$$

Calculate a prediction and 95% interval for time 11 where $\mathbf{z}_{11}^\intercal = (1, 11)$.

The prediction is given by

$$x_{10}(1) = \mathbf{z}_{11}^\intercal \hat{\beta} = 1 \times 10 + 11 \times 2 = 32$$

To calculate the standard error we have that

$$Z^\intercal Z = \begin{pmatrix} 10 & 55 \\ 55 & 385 \end{pmatrix} \qquad \text{and} \qquad (Z^\intercal Z)^{-1} = \begin{pmatrix} 0.46667 & -0.06667 \\ -0.06667 & 0.01212 \end{pmatrix}$$

Therefore we have that

$$\mathrm{Var}(x_{10}(1)) = \hat{\sigma}^2(1 + \mathbf{z}_{11}^\intercal (Z^\intercal Z)^{-1} \mathbf{z}_{11}) = 1.4665$$

and the 95% prediction interval is given by

$$
\begin{aligned}
\mathbf{z}_{n+1}^{\intercal}\hat{\boldsymbol{\beta}} &\pm 1.96\sqrt{\hat{\sigma}^2(1+\mathbf{z}_{n+1}^{\intercal}(Z^{\intercal}Z)^{-1}\mathbf{z}_{n+1})} \\
32 &\pm 1.96\times\sqrt{1.4665} \\
32 &\pm 2.374 \\
(29.63 &, \quad 34.37)
\end{aligned}
$$

Non-linear regression

There are cases where linear regression does not adequately describe the trend in the data but a non-linear functional form is more suitable. The simplest way to model a non-linear relationship is to transform the forecast variable and/or the predictor variable before fitting the regression model. While this allows a non-linear functional form, the model is still linear in the parameters. The most popular transformation is the (natural) logarithm. A log-log functional form is specified as follows:

$$\log(X_t) = \beta_0 + \beta_1 \log(t) + \epsilon_t$$

Other forms of transformations can also be specified; *e.g.* the log-linear form implies only transforming the forecast variable. Note that in order to perform a logarithmic transformation to a variable, all of its observed values must be greater than zero. If the variable X contains zeros, we use the transformation $\log(X+1)$; that is we add one to the value of the variable then take logarithms. This has a similar effect to taking logarithms but avoids the problem of zeros. It also has the neat side-effect of zeros on the original scale remaining zeros on the transformed scale. The log transformation is one special case and the most commonly used of the Box-Cox transformation of the form $(X^{\lambda}-1)/\lambda$ when setting $\lambda = 0$.

There are cases where transforming the data may not be adequate and a more general specification for the functional form describing the trend of the forecast variable is required. The more general model we can use is:

$$X_t = f(t) + \epsilon_t$$

where f is a flexible non-linear function of the time t. One of the simplest ways of fitting a non-linear trend is using quadratic or higher order trend. However, forecasting values outside the range of historical data using these higher order polynomials is often unrealistic and not recommended. A better approach is to make f a piecewise function. One of the simplest specifications is to make f a piecewise linear trend, by introducing a set of breakpoints, called knots, at which the slope of f is allowed to change. This can be achieved by letting $t_1 = t$ and introducing the variable t_2 such that:

$$t_2 = (t-\tau)_+ = \begin{cases} 0 & t < \tau \\ (t-c) & t \geq \tau \end{cases}$$

The notation $(t-\tau)_+$ implies the value $t-\tau$ if it is positive and 0 otherwise. This forces the slope of the trend to bend at point/knot τ. If the associated coefficients of t_1 and t_2 are β_1 and β_2, then β_1 is the slope of the trend before time τ, while $\beta_1+\beta_2$ is the slope of the trend after time τ. Additional bends can be included in the relationship by adding further variables of the form $(t-\tau)_+$, so that we have:

$$t_1 = t, \quad t_2 = (t-\tau_1)_+, \quad \ldots, \quad t_k = (t-\tau_{k-1})_+$$

where $\tau_1, \ldots, \tau_{k-1}$ are the knots at which the line should bend. The number and the position of the knots determine the flexibility of the estimated trend but their choice can be difficult and somewhat arbitrary. A range of automatic knot selection algorithms are available including cross-validation, generalized cross-validation, AIC, … etc. These automatic procedures assume that the data and hence any errors from fitted models are independent. This is an unrealistic assumption in time series which may cause the selection algorithm to break down. For practical purposes, taking a subjective approach and using judgment in selecting the number and the position of knots is a reasonable alternative.

Piecewise linear trends constructed in the above way are a special case of regression splines. A smoother result can be obtained using piecewise cubic spline, where the polynomial segments are piecewise cubics constrained to be continuous and smooth at the knots. A cubic regression spline is defined by setting:

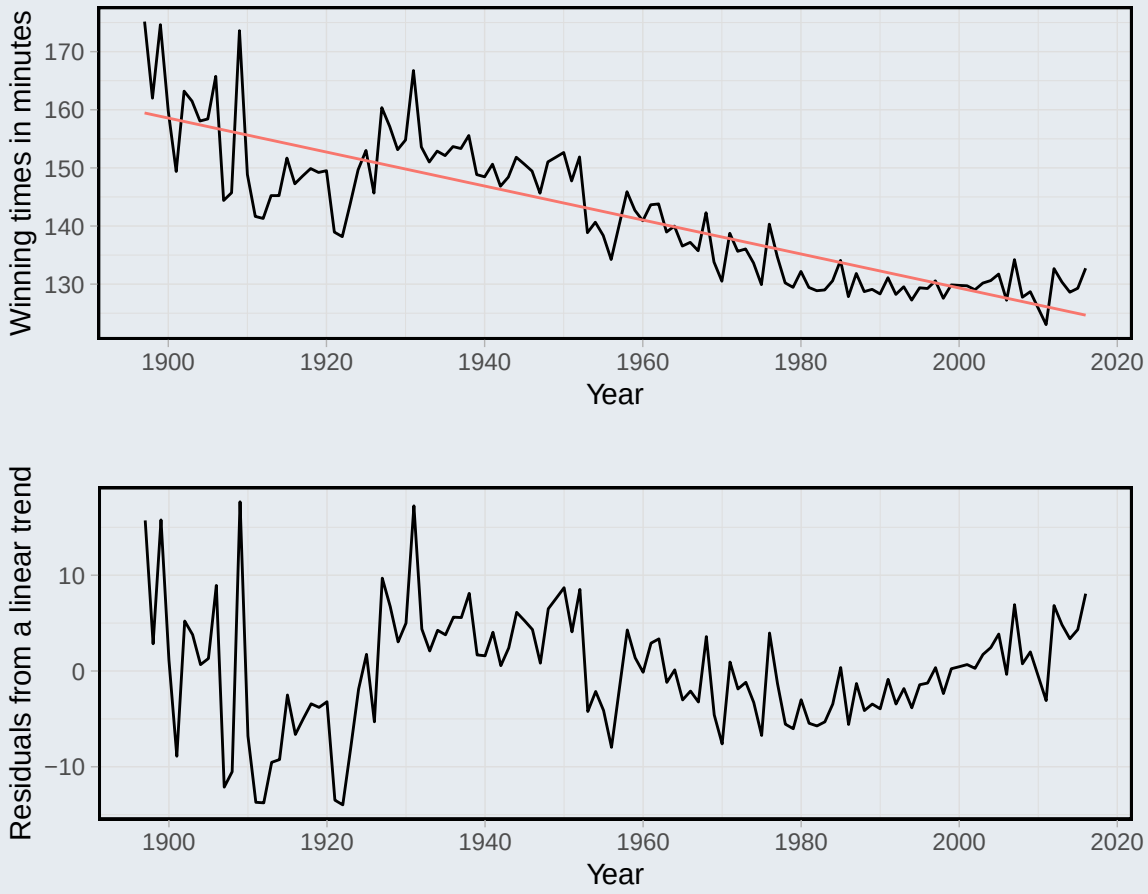
$$t_1 = t, \quad t_2 = t^2, \quad t_3 = t^3, \quad t_4 = (t-c_1)_+, \quad \ldots, t_k = (t-c_{k-3})_+$$

Cubic splines usually give a better fit for the data. However, forecasts of the series become unreliable when t is outside the range of historical data.

Example 10.

The top panel of the figure below shows the Boston marathon winning times (in minutes) since it started in 1897. The time series exhibits a general downward trend implying that the winning times have been improving over the years. Here, we used the function `tslm` in the package `forecast` to fit a linear trend to the time series. This is very similar to using the function `lm` as in the previous example of air traffic data. The bottom panel of the figure below shows the residuals resulting from fitting the linear trend to the data. The plot indicates a non-linear pattern which has not been captured by the linear trend model. There is also some heteroscedasticity, with decreasing variation over the years.

```
library(fpp2)
library(forecast)

fit.lin <- tslm(marathon~trend)

marathon_lin <- autoplot(marathon)+
  autolayer(fitted(fit.lin))+
  ylab("Winning times in minutes") + xlab("Year") + theme(legend.position = "none")

marathon_res <- autoplot(residuals(fit.lin))+
  ylab("Residuals from a linear trend")+xlab("Year")

grid.arrange(marathon_lin,marathon_res,nrow=2)
```

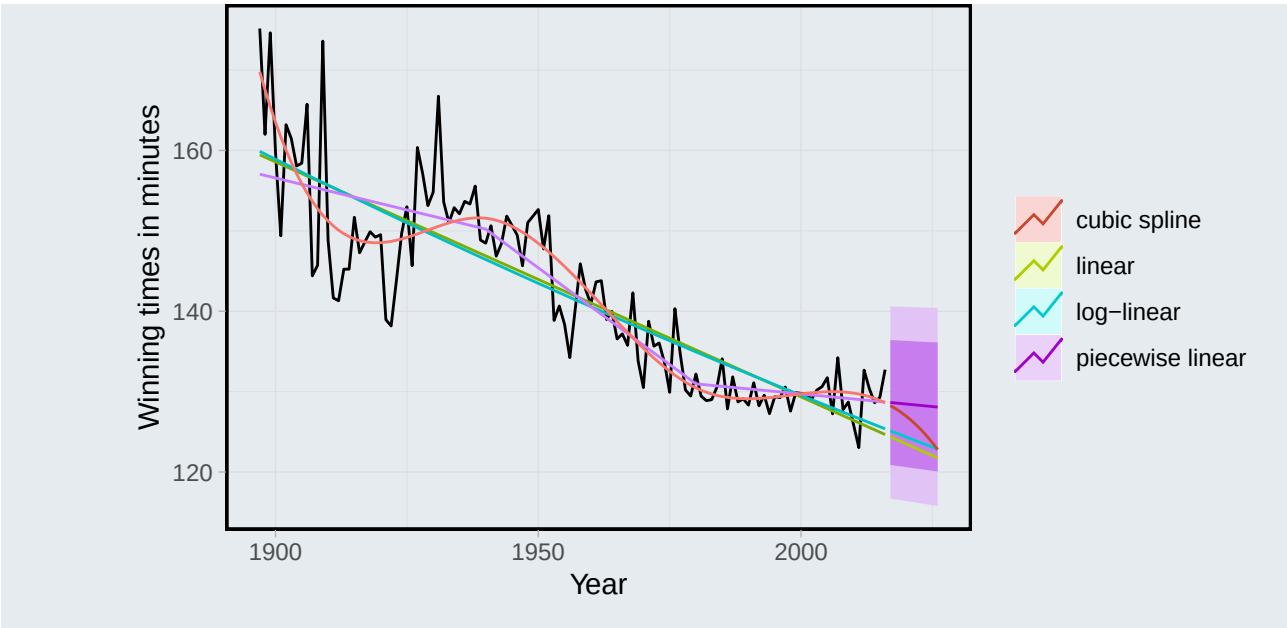


To address the heteroscedasticity in the residuals, a log-linear model is fitted to the data. The fitted trend is shown in the figure below. Although the log-linear model does not seem to fit the data much better than the linear trend, it gives a more sensible projection that the winning times will decrease in the future but at a decreasing rate than a fixed linear rate. The log-linear model is simply fitted using the `tslm()` function by adding the argument `lambda=0`. `lambda` here is the Box-Cox transformation parameter set equal to 0 for a

log transformation.

It can be seen from the above plot that the winning times witness three different periods. There is a lot of variability in the winning times up until 1940, with the winning times decreasing overall but with significant increases during the 1920s. After 1940 there is a near-linear decrease in times, followed by a flattening out after the 1980s, with the suggestion of an upturn towards the end of the sample period. To account for these changes, we specify the years 1940 and 1980 as knots. It is important to note here that this subjective identification of knots can lead to over-fitting, which can be detrimental to the forecast performance of a model, and should be performed with caution.

```
fit.log <- tslm(marathon~trend, lambda=0)

fcasts.lin <- forecast(fit.lin,h=10)
fcasts.log <- forecast(fit.log,h=10)

t <- time(marathon)
t.break1 <- 1940
t.break2 <- 1980

# create the extra variables for the piecewise fit
tb1 <- ts(pmax(0,t-t.break1),start=1897)
tb2 <- ts(pmax(0,t-t.break2),start=1897)
# fit piecewise linear trend (linear regression spline)
fit.pw <- tslm(marathon~ t+tb1+tb2)
# fit cubic regression spline
fit.spline <- tslm(marathon~t+I(t^2)+I(t^3)+I(tb1^3)+I(tb2^3))
# Define the new data for forecasting
t.new <- t[length(t)]+seq(10)
tb1.new <- tb1[length(tb1)]+seq(10)
tb2.new <- tb2[length(tb2)]+seq(10)
newdata <- as.data.frame(cbind(t=t.new,tb1=tb1.new,tb2=tb2.new))

fcasts.pw <- forecast(fit.pw,newdata=newdata)
fcasts.spline <- forecast(fit.spline,newdata=newdata)

autoplot(marathon)+
  autolayer(fitted(fit.lin), series="linear")+
  autolayer(fitted(fit.log), series="log-linear")+
  autolayer(fitted(fit.pw), series = "piecewise linear")+
  autolayer(fitted(fit.spline), series = "cubic spline")+
  autolayer(fcasts.pw, series = "piecewise linear")+
  autolayer(fcasts.lin, series="linear", PI=FALSE)+
  autolayer(fcasts.log,series="log-linear", PI=FALSE)+
  autolayer(fcasts.spline, series = "cubic spline", PI=FALSE)+
  ylab("Winning times in minutes") + xlab("Year") +
  guides(colour=guide_legend(title=""))
```

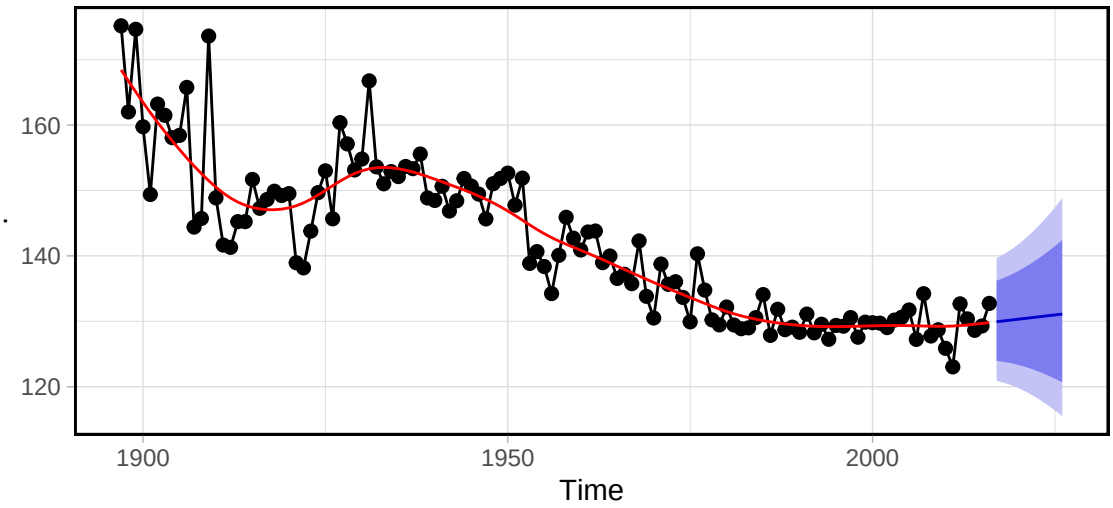


The figure above shows the fitted curves and the forecasts from linear, log-linear, piecewise linear and cubic regression spline trends. The piecewise linear trend provides the best forecasts, while the cubic spline provides the best fit to the historical data but poor forecasts. An alternative formulation of the cubic regression splines, called natural cubic smoothing splines, imposes some constraints such that the spline function is linear at the end. These natural cubic splines usually give better forecasts without compromising the fit. The natural cubic spline uses more knots (often a knot at each observation) than the cubic regression spline but the coefficients are constrained through a penalty term, added to the least squares objective function, to prevent overfitting. This roughness penalty ensures that the estimated curve captures the curvature of the data without interpolating the observations. A common choice is to put a penalty on the second derivative of the fitted curve as follows:

$$(X_t - f(t))^2 + \delta \int_0^T [f''(t)]^2 dt$$

We then choose $f(.)$ that minimises this penalised sum of squares. The trade-off between the model fit and the model smoothness is controlled by the smoothing parameter δ and not the number of knots. This has the added advantage that knot selection is not subjective. Here we use the function `splinef` in the `forecast` package to produce the natural cubic spline forecasts. We can also use a log transformation by setting `lambda=0` to handle the heteroscedasticity in the data as follows:

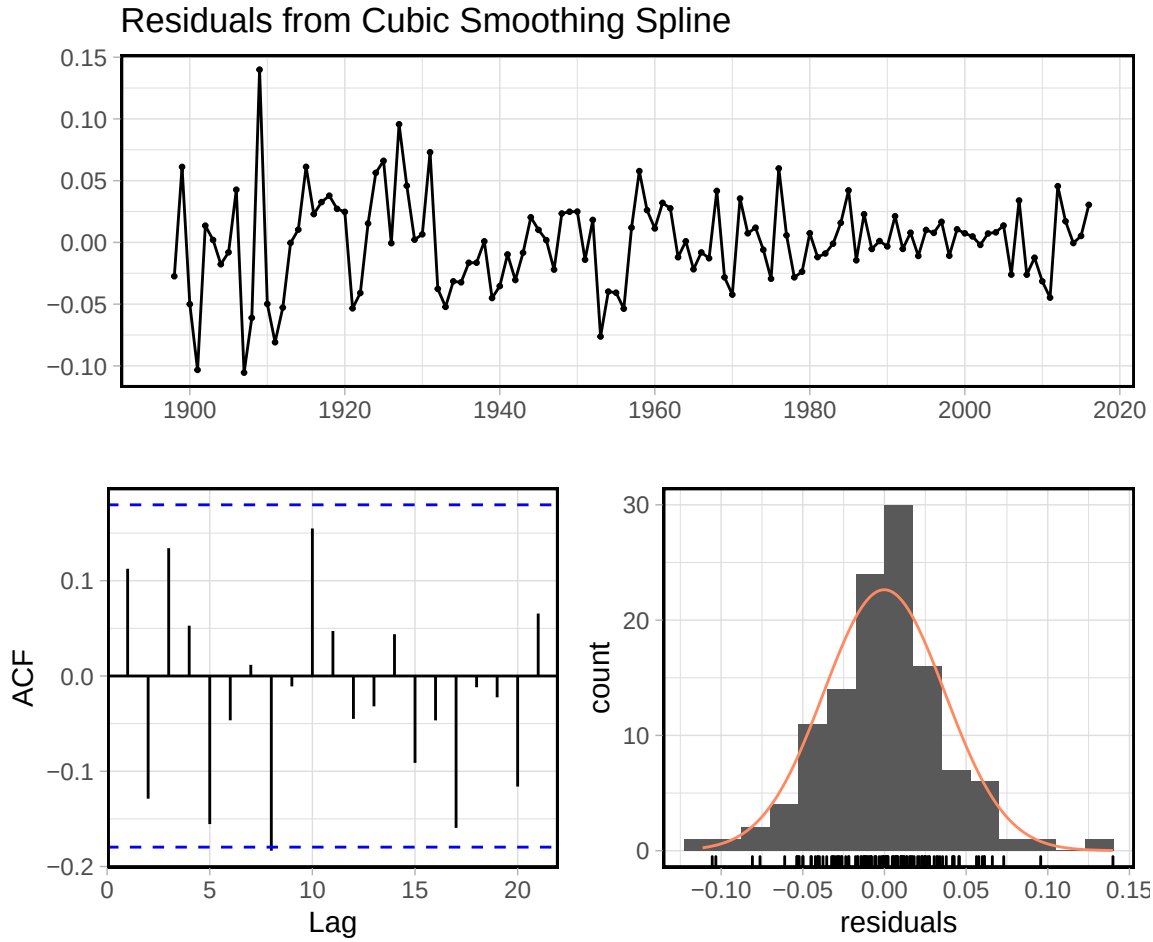
```
marathon %>%
  splinef(lambda=0) %>%
  autoplot()
```



The dark and light blue areas represent the 80% and 95% prediction intervals respectively. You can choose to draw only the 95% intervals by adding the argument `level=95` to the function `splinef`.

The residuals from the natural cubic smoothing spline appear to capture the trend well, although there is some heteroscedasticity remaining. The wide prediction interval associated with the forecasts reflects the variability observed in the historical winning times.

```
marathon %>%
  splinef(lambda=0) %>%
  checkresiduals()
```



Task 7.

The file `globaltemp.csv` gives the global temperature anomaly (annual-mean surface air temperature deviation from the 1951-1980 mean) obtained from meteorological station data. Obtain a forecast for the next 30 years for the data in the `NoSmoothing` column using linear and cubic regression splines and compare the results.

Exponential smoothing

Exponential smoothing is another simple procedure that does not assume a parametric model for the data, and is similar to moving average smoothing discussed in Week 6. It makes one-step ahead forecasts of the form

$$\hat{x}_n(1) = c_0 x_n + c_1 x_{n-1} + c_2 x_{n-2} + \ldots + c_{n-1} x_1$$

where the coefficients $(c_0, \ldots, c_{n-1})$ are called weights and must sum to one so that the prediction is of the correct size. In addition, the weights decrease as the observations move further away from the time point being predicted, i.e $c_0 \geq c_1 \geq \ldots \geq c_{n-1}$. The weights used in exponential smoothing are as follows.

Definition 3.

*Given data $(x_1, \ldots, x_n)$, the one-step ahead forecast using **exponential smoothing** is given by*

$$\hat{x}_n(1) = \alpha x_n + \alpha(1-\alpha)x_{n-1} + \alpha(1-\alpha)^2 x_{n-2} + \ldots + \alpha(1-\alpha)^{n-1}x_1$$

where $\alpha \in [0, 1]$ is a smoothing parameter.

Notes

- If α is close to one, predictions are based on only the last few observations.
- If α is close to zero, predictions are based on a large number of previous observations.
- If the series has infinite length then the weights sum to 1 as required. To see this note that $c_i = \alpha(1-\alpha)^i$ and

$$\begin{aligned}
\sum_{i=0}^{\infty} c_i &= \sum_{i=0}^{\infty} \alpha(1-\alpha)^i \\
&= \alpha \sum_{i=0}^{\infty} (1-\alpha)^i \\
&= \alpha \times \frac{1}{1-(1-\alpha)} \\
&= 1
\end{aligned}$$

For finite n, the sum of the coefficients is approximately 1 because for large enough n, $c_n = \alpha(1-\alpha)^n \approx 0$.

The one-step ahead forecast can be written recursively as follows

$$\begin{aligned}
\hat{x}_n(1) &= \alpha x_n + \alpha(1-\alpha)x_{n-1} + \alpha(1-\alpha)^2 x_{n-2} + \ldots + \alpha(1-\alpha)^{n-1}x_1 \\
&= \alpha x_n + (1-\alpha)[\alpha x_{n-1} + \alpha(1-\alpha)x_{n-2} + \ldots + \alpha(1-\alpha)^{n-2}x_1] \\
&= \alpha x_n + (1-\alpha)\hat{x}_{n-1}(1)
\end{aligned}$$

making it straightforward computationally to update the forecasts in light of new data. To start the process, we set $\hat{x}_1(1) = x_2$.

Choosing α

Calculate the root mean square prediction error for a range of α values, and choose the one that minimises this quantity. That is, for each candidate value of α

- Calculate $\hat{x}_1(1), \ldots, \hat{x}_{n-1}(1)$ using the recursive formula described above.
- Calculate the root mean square prediction error

$$\text{RMSPE} = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n-1}(x_{k+1} - x_k(1))^2}$$

Then choose the value of α that minimises this quantity.

Measuring uncertainty

For exponential smoothing it has been shown that an approximate 95% prediction interval for $x_n(1)$ is given by

$$\hat{x}_n(1) \pm 1.96\sqrt{\text{Var}(e_n(1))}$$

where $\text{Var}(e_n(1))$ can be approximated as the variance of the forecast errors $e_1(1), e_2(1), \ldots, e_{n-1}(1)$, i.e

$$\text{Var}(e_n(1)) = \frac{1}{n-2}\sum_{i=1}^{n-1}(e_i(1) - \bar{e})^2$$

with $\bar{e} = \sum_{i=1}^{n-1} e_i(1)/(n-1)$.

Example 11.

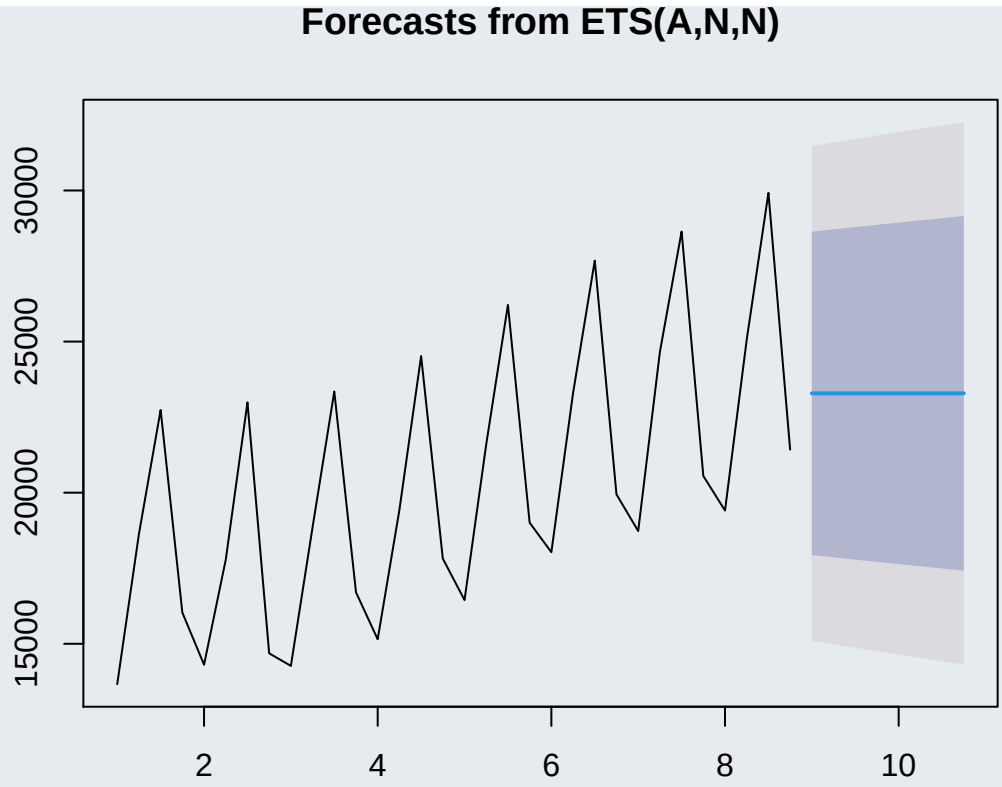
The one-step ahead forecast for the air traffic data is given below. The prediction and uncertainty interval from the regression approach is given by 20818.29 (19235.76, 22400.83), which is markedly different from the values given below for exponential smoothing. Note also that the uncertainty intervals for exponential smoothing are much wider.

```
predict <- ses(x, h=1, level=95, fan=FALSE)
summary(predict)


Forecast method: Simple exponential smoothing

Model Information:
Simple exponential smoothing

Call:
 ses(y = x, h = 1, level = 95, fan = FALSE)

  Smoothing parameters:
    alpha = 0.17

  Initial states:
    l = 17749.1463

  sigma:  4175.187

     AIC     AICc      BIC
648.4008 649.2580 652.7980

Error measures:
                    ME     RMSE      MAE      MPE     MAPE      MASE         ACF1
Training set 1017.934 4042.607 3507.958 1.573716 16.78715 0.7534899 -0.04362285

Forecasts:
   Point Forecast    Lo 95    Hi 95
33       23287.03 15103.82 31470.25
```

In the above code we used the function `ses` from `library(forecast)`, which is appropriate for stationary time series. If the series is non-stationary we can use a more sophisticated version of exponential smoothing, called a *triple exponential model* or *Holt-Winters exponential smoothing* which can be fit using the function `ets()` from `library(forecast)`. We will not go into detail about the technical aspects of this model, but we will see how it can be applied to the air traffic example.

First of all, let us note that using `ets(ts,model="ANN")` is the same as using `ses(ts)` because we are specifying an additive error (first letter is "A"), no trend (second letter is "N") and no seasonality (third letter is "N"):

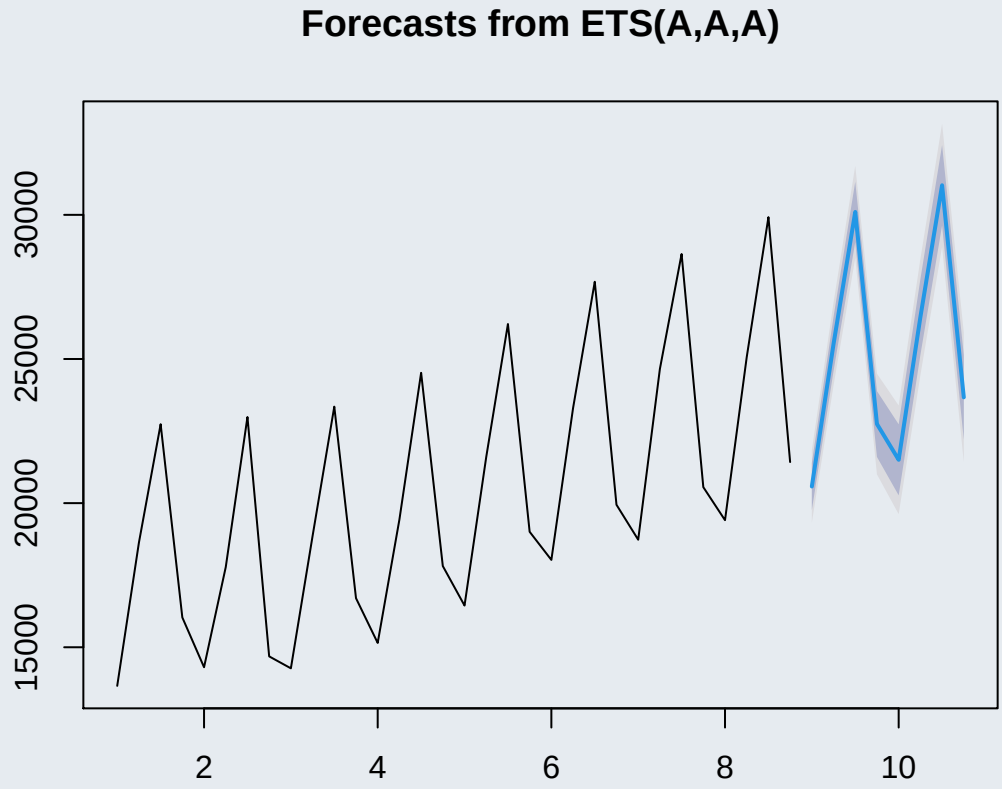
```
predict1 <- ets(ts(x, freq=4),model="ANN")
f1 <- forecast(predict1)
plot(f1)
```



Here the blue dots show the point estimates, the light grey band gives the 80% confidence interval and the dark grey band gives the 95% confidence interval.

Note in the above code the use of `ts(x, freq=4)` because we have quarterly data. To allow for additive trend and seasonality we use:

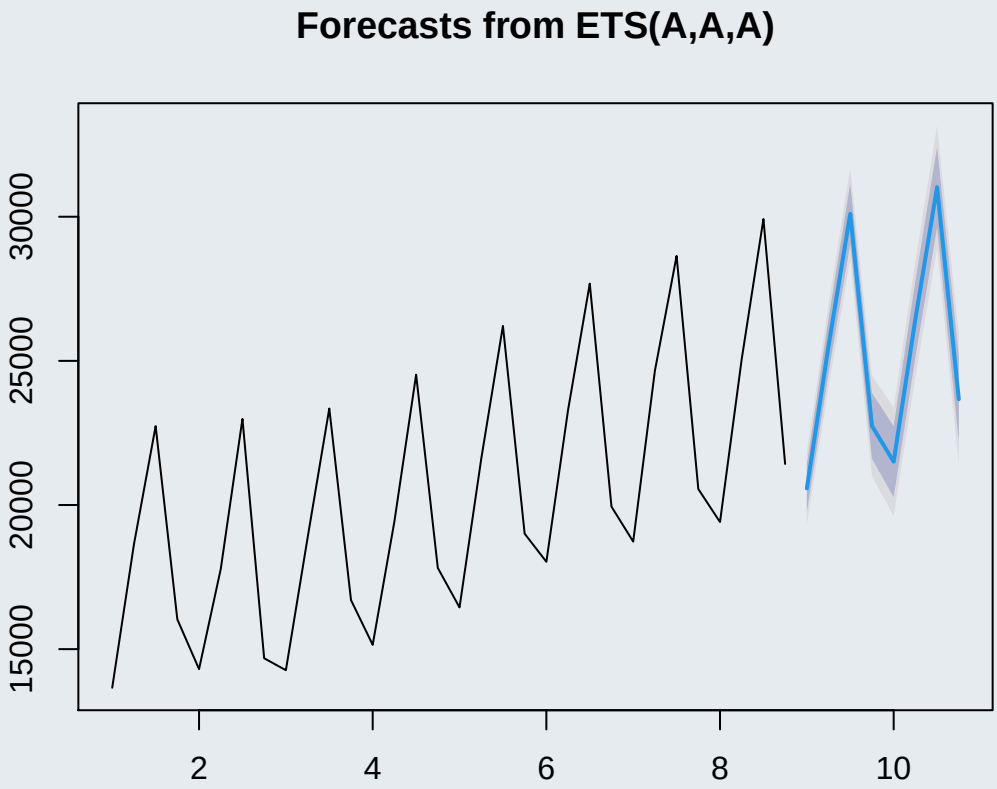
```
predict2 <- ets(ts(x, frequency=4),model="AAA")
f2 <- forecast(predict2)
plot(f2)
```



This forecast seems more appropriate for the air traffic data.

Note that we can use the `ets()` function without specifying the type of model, but rather letting it automatically select a best-fitting model according to some criterion. Applying this to the air traffic data we get:

```
predict3 <- ets(ts(x, frequency=4))
f3 <- forecast(predict3)
plot(f3)
```



In this case it chooses the same model as above.

Forecasting from AR(p) models

The remainder of this section describes how to forecast from ARIMA models, beginning with an AR(p) model. We assume the time series being predicted is stationary with zero mean, as any trend or seasonal variation can be predicted using the regression methods described above. It can be shown that given a time series $(x_1, \ldots, x_n)$, the k-step ahead forecast that is optimal in a mean square error sense is:

$$\hat{x}_n(k) = E(X_{n+k}|X_n, X_{n-1}, \ldots, X_1)$$

the conditional expectation of X_{n+k} given the existing values of the series. Forecasting using regression methods also uses exactly this conditional expectation.

AR(1) forecasting

Suppose the time series $(x_1, \ldots, x_n)$ is represented by an AR(1) process $X_t = \alpha X_{t-1} + Z_t$. Then the one-step ahead forecast is given by

$$\begin{aligned}
\hat{x}_n(1) &= E(X_{n+1}|X_n, X_{n-1}, \ldots, X_1) \\
&= E(\alpha X_n + Z_{n+1}|X_n, X_{n-1}, \ldots, X_1) \\
&= \alpha E(X_n|X_n, X_{n-1}, \ldots, X_1) + E(Z_{n+1}|X_n, X_{n-1}, \ldots, X_1) \\
&= \alpha x_n
\end{aligned}$$

where x_n is the observed value of the series at time n and $E(Z_{n+1}|X_n, X_{n-1}, \ldots, X_1) = 0$. The two-step ahead forecast is given by

$$\begin{aligned}\hat{x}_n(2) &= E(X_{n+2}|X_n, X_{n-1}, \ldots, X_1) \\ &= E(\alpha X_{n+1} + Z_{n+2}|X_n, X_{n-1}, \ldots, X_1) \\ &= \alpha E(X_{n+1}|X_n, X_{n-1}, \ldots, X_1) + E(Z_{n+2}|X_n, X_{n-1}, \ldots, X_1) \\ &= \alpha^2 x_n\end{aligned}$$

Iterating the above procedure gives the k-step ahead forecast as

$$\hat{x}_n(k) = \alpha^k x_n$$

The forecast error variance at one step ahead is given by

$$\begin{aligned}\mathrm{Var}(e_n(1)) &= \mathrm{var}(X_{n+1} - \hat{x}_n(1)) \\ &= \mathrm{Var}(\alpha X_n + Z_{n+1} - \alpha X_n) \\ &= \mathrm{Var}(Z_{n+1}) \\ &= \sigma_z^2\end{aligned}$$

and at two steps ahead it is

$$\begin{aligned}\mathrm{Var}(e_n(2)) &= \mathrm{Var}(X_{n+2} - \hat{x}_n(2)) \\ &= \mathrm{Var}(\alpha X_{n+1} + Z_{n+2} - \alpha \hat{x}_n(1)) \\ &= \mathrm{Var}(Z_{n+2}) + \alpha^2 \mathrm{Var}(X_{n+1} - \hat{x}_n(1)) \\ &= \sigma_z^2(1 + \alpha^2)\end{aligned}$$

This process can also be iterated to give

$$\mathrm{Var}(e_n(k)) = \sigma_z^2(1 + \alpha^2 + \ldots + \alpha^{2k}) = \sigma_z^2 \frac{1 - \alpha^{2k}}{1 - \alpha^2}$$

because it is the sum of a geometric progression with finitely many terms. Approximate 95% prediction intervals are now straightforward to calculate as

$$\hat{x}_n(k) \pm 1.96\sqrt{\mathrm{Var}(e_n(k))}$$

AR(p) forecasting

For an AR(p) process $X_t = \alpha_1 X_{t-1} + \ldots + \alpha_p X_{t-p} + Z_t$, the one-step ahead forecast is given by

$$\begin{aligned}\hat{x}_n(1) &= E(X_{n+1}|X_n, X_{n-1}, \ldots, X_1) \\ &= E(\alpha_1 X_n + \ldots + \alpha_p X_{n-p+1} + Z_{n+1}|X_n, X_{n-1}, \ldots, X_1) \\ &= \alpha_1 x_n + \ldots + \alpha_p x_{n-p+1}\end{aligned}$$

Then for any k, the k-step ahead forecast is given by

$$\begin{aligned}\hat{x}_n(k) &= E(X_{n+k}|X_n, X_{n-1}, \ldots, X_1) \\ &= E(\alpha_1 X_{n+k-1} + \ldots + \alpha_p X_{n+k-p} + Z_{n+k}|X_n, X_{n-1}, \ldots, X_1) \\ &= \alpha_1 E(X_{n+k-1}|X_n, X_{n-1}, \ldots, X_1) + \ldots + \alpha_p E(X_{n+k-p}|X_n, X_{n-1}, \ldots, X_1)\end{aligned}$$

Two cases occur for these conditional expectations

1. If X_{n+k-j} has been observed, then $E(X_{n+k-j}|X_n, X_{n-1}, \ldots, X_1)$ is equal to its observed value, x_{n+k-j}.
2. If X_{n+k-j} is a future value, then $E(X_{n+k-j}|X_n, X_{n-1}, \ldots, X_1)$ has already been forecast as one of $\hat{x}_n(1), \ldots, \hat{x}_n(k-1)$.

The error variance for the k-step ahead forecast has the general form

$$\mathrm{Var}(e_n(k)) = \sigma_z^2 \sum_{i=0}^{k-1} \theta_i^2$$

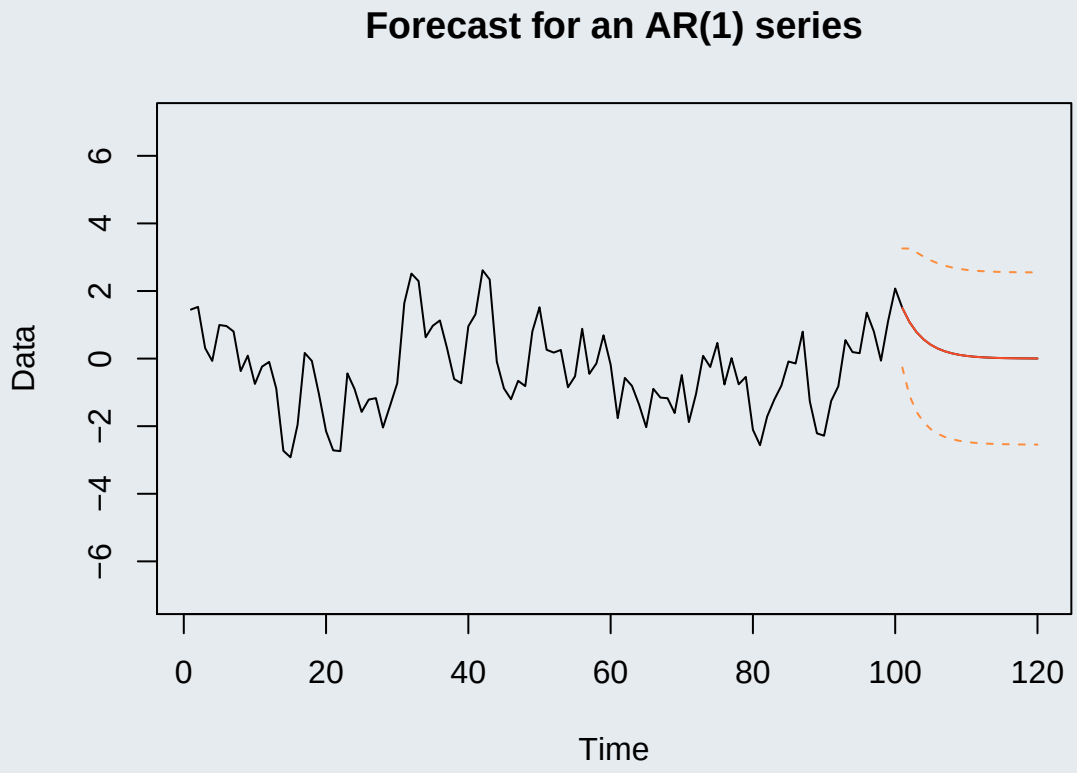
where $\theta_0 = 1$, and the remaining θ_i are algebraically nasty to determine. Prediction intervals can be calculated using the same formula as for the AR(1) model.

Example 12.

The graph below shows a simulated AR(1) process of length 100, with 20 future predictions together with prediction intervals. Note how the forecasts fall to zero as k increases from 1 to 20.

The R code to implement this prediction is given below.

```
x <- arima.sim(model=list(ar=0.8), n=100)
model <- arima(x, order=c(1,0,0), include.mean=FALSE)
predict.ar1 <- predict(model, n.ahead = 20, se.fit = TRUE)
predict.lci <- predict.ar1$pred - 1.96*predict.ar1$se
predict.uci <- predict.ar1$pred + 1.96*predict.ar1$se

plot(1:120, c(x, predict.ar1$pred), type="l", xlab="Time", ylab="Data",
      ylim=c(-7,7), main="Forecast for an AR(1) series")
lines(101:120, predict.ar1$pred, col="#fc4e2a")
lines(101:120, predict.lci, lty=2, col="#fd8d3c")
lines(101:120, predict.uci, lty=2, col="#fd8d3c")
```



Example 13.

Consider an AR(1) time series process $X_t = 0.9X_{t-1} + Z_t$, where $\hat{\sigma}_z^2 = 1$ and $x_n = 20$. Let us calculate the

one- and two-step ahead forecasts and the associated error variances.

The forecasts are given by

$$\hat{x}_n(1) = \alpha x_n = 0.9 \times 20 = 18$$

$$\hat{x}_n(2) = \alpha^2 x_n = 0.9^2 \times 20 = 16.2$$

and the error variances are

$$\text{Var}(e_n(1)) = \hat{\sigma}_z^2 = 1 \qquad \text{and} \qquad \text{Var}(e_n(2)) = \hat{\sigma}_z^2(1+\alpha^2) = 1.81$$

Task 8.

Consider an AR(1) time series process $X_t = 0.1X_{t-1} + Z_t$, where $\hat{\sigma}_z^2 = 1$ and $x_n = 20$. Calculate the one- and two-step ahead forecasts and the associated error variances.

Notes

- As the lag 1 coefficient gets smaller the forecasts get closer to zero.
- As the lag 1 coefficient gets smaller the two-step ahead forecast error gets smaller.

Forecasting from MA(q) models

Forecasting with an MA(q) model is similar to forecasting with an AR(p) model, as the former is also based on the conditional expectation

$$\hat{x}_n(k) = E(X_{n+k}|X_n, X_{n-1}, \ldots, X_1)$$

We begin the section by focusing on an MA(1) model.

MA(1) forecasts

For the MA(1) model $X_t = \lambda Z_{t-1} + Z_t$, the one-step ahead forecast is given by

$$\begin{aligned}
\hat{x}_n(1) &= E(X_{n+1}|X_n, X_{n-1}, \ldots, X_1) \\
&= E(\lambda Z_n + Z_{n+1}|X_n, X_{n-1}, \ldots, X_1) \\
&= \lambda E(Z_n|X_n, X_{n-1}, \ldots, X_1) + E(Z_{n+1}|X_n, X_{n-1}, \ldots, X_1) \\
&= \lambda z_n
\end{aligned}$$

The last line is true because

- $X_n, X_{n-1}, \ldots, X_1$ do not depend on Z_{n+1}, and hence

$$E(Z_{n+1}|X_n, X_{n-1}, \ldots, X_1) = E(Z_{n+1}) = 0$$

- In contrast, X_n depends on Z_n so

$$E(Z_n|X_n, X_{n-1}, \ldots, X_1) \neq E(Z_n) = 0$$

Z_n cannot be observed directly but it can be estimated as follows. Re-write the MA(1) process as, $Z_t = X_t - \lambda Z_{t-1}$ and assuming that $Z_0 = 0$, Z_t can be estimated iteratively from $t = 1, \ldots, n$ by replacing X_t by its observed value x_t.

For $k > 1$ the k-step ahead forecast is given by

$$\begin{aligned}\hat{x}_n(k) &= E(X_{n+k}|X_n, X_{n-1}, \ldots, X_1) \\ &= E(\lambda Z_{n+k-1} + Z_{n+k}|X_n, X_{n-1}, \ldots, X_1) \\ &= 0\end{aligned}$$

The forecast error variance at one-step ahead is given by

$$\begin{aligned}\text{Var}(e_n(1)) &= \text{Var}(X_{n+1} - \hat{x}_n(1)) \\ &= \text{Var}(\lambda Z_n + Z_{n+1} - \lambda Z_n) \\ &= \text{Var}(Z_{n+1}) \\ &= \sigma_z^2\end{aligned}$$

while for $k > 1$ it is given by

$$\begin{aligned}\text{Var}(e_n(k)) &= \text{Var}(X_{n+k} - \hat{x}_n(k)) \\ &= \text{Var}(X_{n+k}) \\ &= \sigma_z^2(1 + \lambda^2)\end{aligned}$$

Then 95% prediction intervals can be calculated as before using the formula

$$\hat{x}_n(k) \pm 1.96\sqrt{\text{Var}(e_n(k))}$$

MA(q) forecasts

Forecasts from an MA(q) model work in the same way as those from an MA(1) model. The one-step ahead forecast is given by

$$\begin{aligned}\hat{x}_n(1) &= E(X_{n+1}|X_n, X_{n-1}, \ldots, X_1) \\ &= E(\lambda_1 Z_n + \ldots + \lambda_q Z_{n-q+1} + Z_{n+1}|X_n, X_{n-1}, \ldots, X_1) \\ &= \lambda_1 z_n + \ldots + \lambda_q z_{n-q+1}\end{aligned}$$

where as before the current and past values of Z_t are calculated recursively from the MA(q) equation

$$Z_t = X_t - \lambda_1 Z_{t-1} - \ldots - \lambda_q Z_{t-q}$$

with the initial conditions $Z_0 = Z_{-1} = \ldots = Z_{-q+1} = 0$. The general k-step ahead forecast is calculated in an identical way, where current and past values of Z_t are estimated from the data, while future values are set to zero. Therefore the forecast is given by

$$\hat{x}_n(k) = \begin{cases}\lambda_k z_n + \cdots + \lambda_q z_{n+k-q} & \text{if } k \leq q \\ 0 & \text{if } k > q.\end{cases}$$

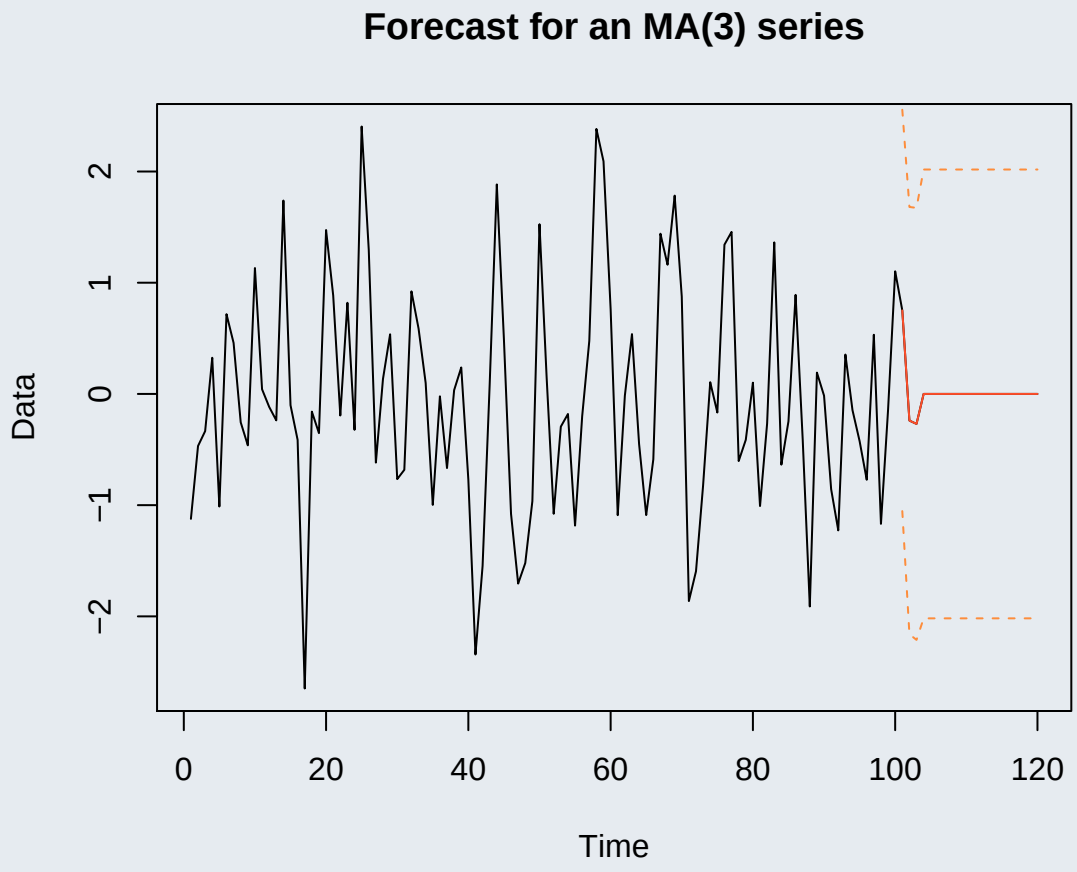
It is straightforward to show that the k-step ahead error variance is given by

$$e_n(k) = \begin{cases}\sigma_z^2[1 + \sum_{i=1}^{k-1} \lambda_i^2] & \text{if } k \leq q \\ \sigma_z^2[1 + \sum_{i=1}^{q} \lambda_i^2] & \text{if } k > q.\end{cases}$$

Example 14.

The graph below shows a simulated MA(3) process of length 100, with 20 future predictions together with prediction intervals. Note how the forecasts fall to zero for $k > 3$, which was shown algebraically earlier.

```
x <- arima.sim(model=list(ma=0.4, 0.5, 0.4), n=100)

model <- arima(x, order=c(0,0,3), include.mean=FALSE)

# Prediction
predict.ma3 <- predict(model, n.ahead = 20, se.fit = TRUE)
predict.lci <- predict.ma3$pred - 1.96*predict.ma3$se
predict.uci <- predict.ma3$pred + 1.96*predict.ma3$se

# Plotting
plot(1:120, c(x, predict.ma3$pred), type="l", xlab="Time", ylab="Data",
      main="Forecast for an MA(3) series")
lines(101:120, predict.ma3$pred, col="#fc4e2a")
lines(101:120, predict.lci, lty=2, col="#fd8d3c")
lines(101:120, predict.uci, lty=2, col="#fd8d3c")
```



Example 15.

Consider modelling the short time series $\mathbf{x} = (3, 8, 2, 5, 6)$ with an MA(1) time series process $X_t = 0.7Z_{t-1} + Z_t$, where $\hat{\sigma}_z^2 = 1$. To calculate the one- and two-step ahead forecasts $x_5(1)$ and $x_5(2)$ as well as their associated error variances, we need to recursively estimate $Z_1, \ldots, Z_5$, assuming that $z_0 = 0$. This gives

$$z_1 = 3, \quad z_2 = 5.9, \quad z_3 = -2.13, \quad z_4 = 6.491, \quad z_5 = 1.4563$$

$$\hat{x}_5(1) = \lambda z_5 = 0.7 \times 1.4563 = 1.01941$$

$$\hat{x}_n(2) = 0$$

and the error variances are

$$\mathrm{Var}(e_n(1)) = \hat{\sigma}_z^2 = 1 \qquad \text{and} \qquad \mathrm{Var}(e_n(2)) = \hat{\sigma}_z^2(1+\lambda) = 1.49$$

Forecasting time series with trend, seasonality and correlation

There are a number of ways to forecast a time series that contains trend and seasonal variation in addition to short-term correlation. The method we consider in this course is a natural combination of regression and ARMA(p, q) models. For the time series model

$$X_t = m_t + s_t + e_t$$

we represent the trend and seasonal variation by

$$m_t + s_t = \mathbf{z}_t^\top \boldsymbol{\beta}$$

while the residuals are given by

$$e_t^* = X_t - \mathbf{z}_t^\top \hat{\boldsymbol{\beta}}$$

and are modelled by a stationary ARMA(p, q) process. Then the k-step ahead forecast is given by

$$\hat{x}_n(k) = \mathbf{z}_{n+1}^\top \hat{\boldsymbol{\beta}} + e_n^*(k)$$

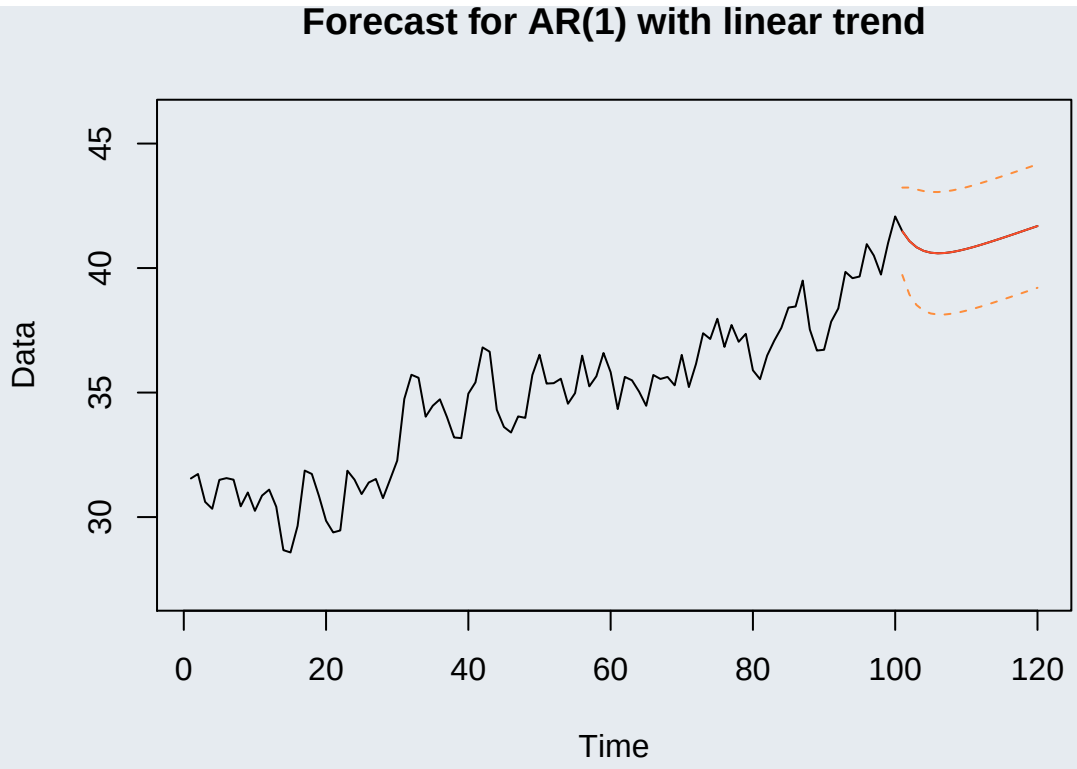
where the stationary process $e_n^*(k)$ is predicted using an AR(p), MA(q) or ARMA(p, q) model.

Example 16.

The graph below shows a simulated AR(1) process with a linear trend of length 100, with 20 future predictions together with prediction intervals.

The R code to implement this prediction is given below.

```
time <- 1:100
time.predict <- 101:120
x <- arima.sim(model=list(ar=0.8), n=100) + 30 + 0.1*time
model.ar1 <- arima(x, order=c(1,0,0), xreg=time, include.mean=TRUE)

# Prediction
predict.ar1 <- predict(model.ar1, n.ahead = 20, newxreg = time.predict,
                       se.fit = TRUE)
ar1.LCI <- predict.ar1$pred - 1.96*predict.ar1$se
ar1.UCI <- predict.ar1$pred + 1.96*predict.ar1$se

# Plotting
plot(1:120, c(x, predict.ar1$pred), type="l", xlab="Time", ylab="Data",
     ylim=c(27,46), main="Forecast for AR(1) with linear trend")
lines(101:120, predict.ar1$pred, col="#fc4e2a")
lines(101:120, ar1.LCI, lty=2, col="#fd8d3c")
lines(101:120, ar1.UCI, lty=2, col="#fd8d3c")
```



Note that the `forecast()` function from `library(forecast)` can also be used to predict from an ARIMA model and that it is also possible to obtain automated ARIMA forecasts using the `auto.arima()` function.

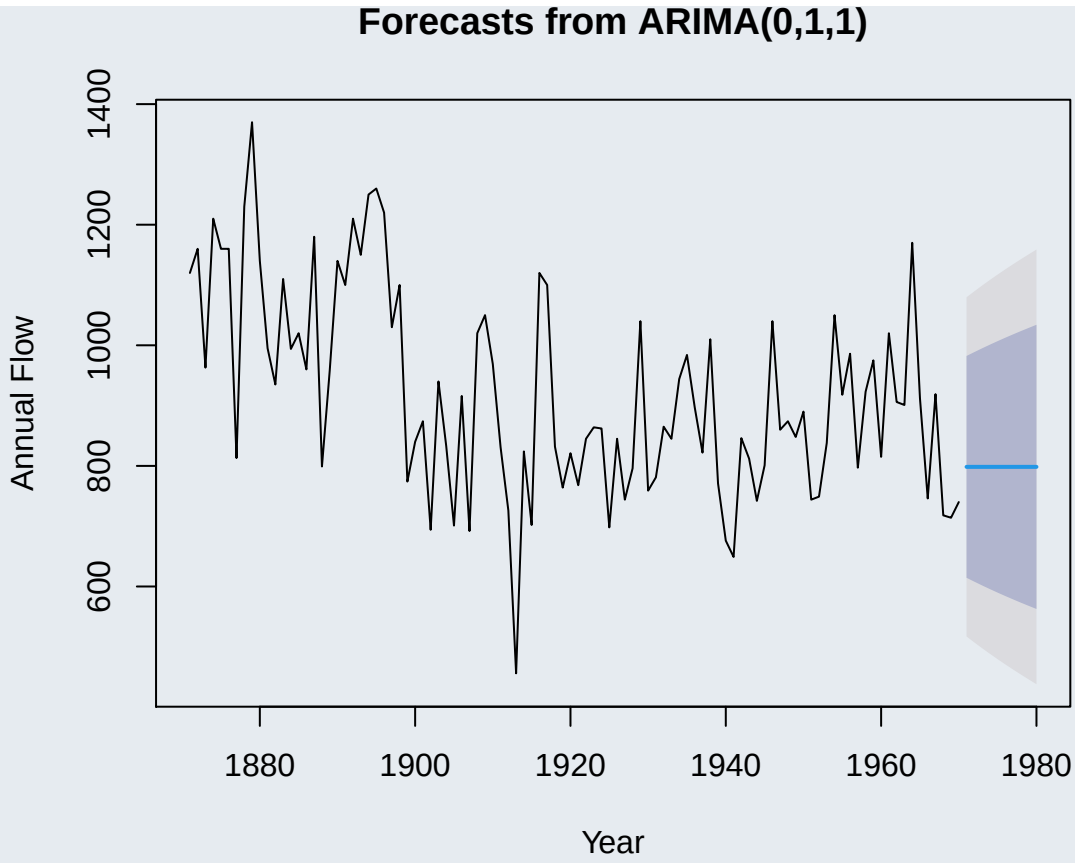
Example 17.

Suppose that we would like to forecast the next few values for the `Nile` time series data from Task 6. We can start by using the ARIMA model fitted in Task 6.

```
library(forecast)
fitNile <- arima(Nile, order=c(0,1,1))
forecast(fitNile, 3)
```

```
     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
1971       798.3673 614.4307 982.3040 517.0605 1079.674
1972       798.3673 607.9845 988.7502 507.2019 1089.533
1973       798.3673 601.7495 994.9851 497.6663 1099.068
```

```
plot(forecast(fitNile, 10), xlab="Year", ylab="Annual Flow")
```



Another possibility is to find the best ARIMA model that optimizes the AIC (or other similar criterion). This can be done by using the `auto.arima()` function. The model chosen is an ARIMA(1,1,1) as shown below.

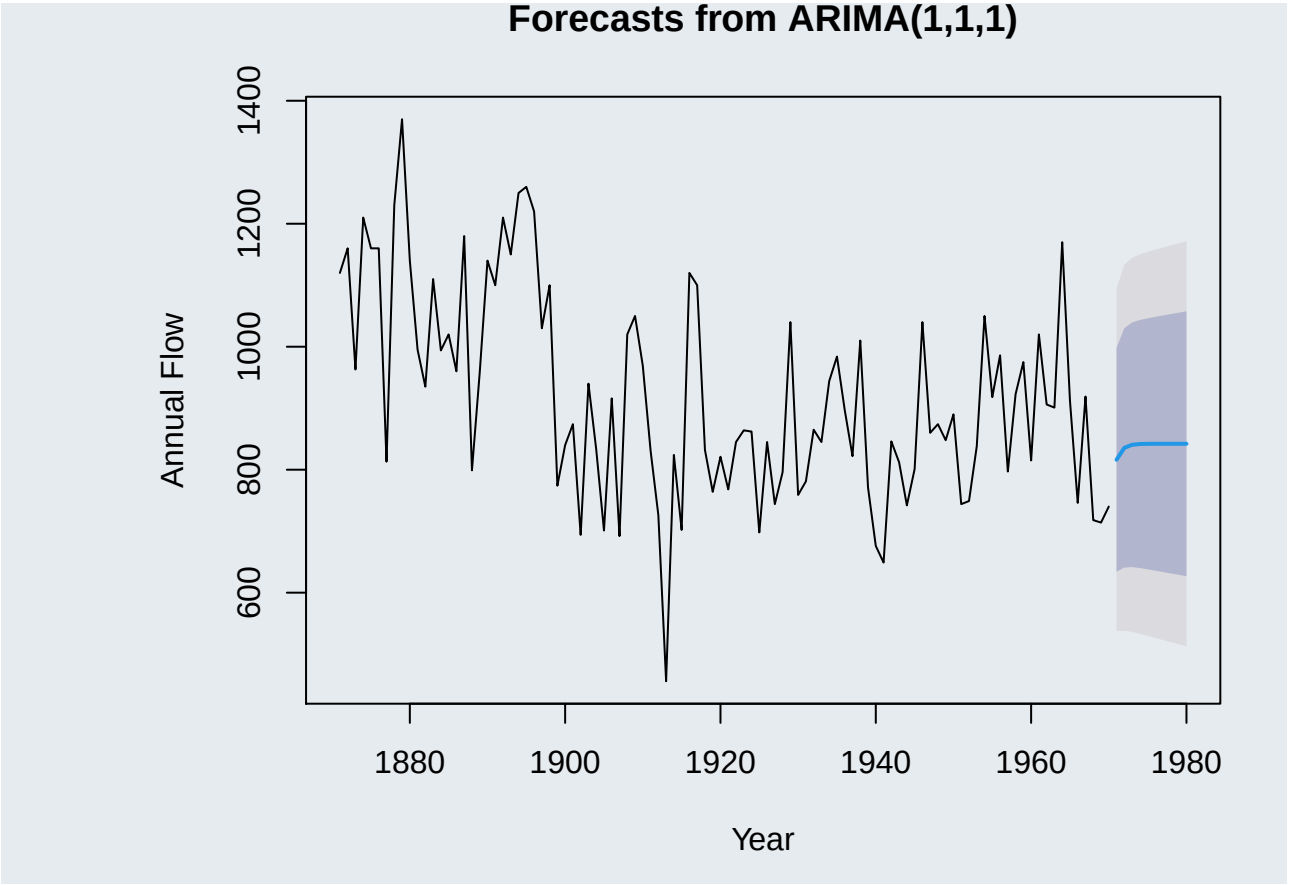
```
fitNile2 <- auto.arima(Nile)
fitNile2
```

```
Series: Nile
ARIMA(1,1,1)

Coefficients:
         ar1      ma1
      0.2544  -0.8741
s.e.  0.1194   0.0605

sigma^2 estimated as 20177:  log likelihood=-630.63
AIC=1267.25   AICc=1267.51   BIC=1275.04
```

```
forecast(fitNile2,3)
```

```
     Point Forecast    Lo 80     Hi 80    Lo 95    Hi 95
1971       816.1813 634.1427  998.2199 537.7773 1094.585
1972       835.5596 640.8057 1030.3136 537.7091 1133.410
1973       840.4889 641.5646 1039.4132 536.2604 1144.717
```

```
plot(forecast(fitNile2, 10), xlab="Year", ylab="Annual Flow")
```



Task 9.

The file `globaltemp.csv` gives the global temperature anomaly (annual-mean surface air temperature deviation from the 1951-1980 mean) obtained from meteorological station data. Obtain a forecast for the next 30 years for the data in the `NoSmoothing` column.

Other forecasting methods

The above forecasting methods fall in the class of classical time series approach for forecasting. Other classical methods include Vector Auto-regression Moving-Average (VARMA) which represent the generalisation of ARMA to multiple parallel time series, i.e. multivariate time series. On the other hand, there is a class of machine learning methods for forecasting, which is out of the scope of this course. This class includes neural networks, kernel and K-Nearest Neighbor regression and Gaussian processes.

Additional resources on ARMA/ARIMA processes and forecasting

Time Series Analysis with Applications in R by Cryer and Chan:

- **Chapter 5**: Models for nonstationary time series
- **Chapter 9**: Forecasting

[**Time Series Analysis and Its Applications: With R Examples by Shumway and Stoffer**]

(https://glasgow.summon.serialssolutions.com/#!/search?bookMark=ePnHCXMw42LgTQStzc4rAe_hSmFm4DIGRrMpaIYPNHEIbJBYAIt
PB465BGfBOylm5qArvkjQgkA7q4zFA): **Chapter 3**: ARIMA Models (includes ARMA, ARIMA and discusses forecasting)

Forecasting: Principles and Practice by Hyndman and Athanasopoulos

More on exponential smoothing methods can be found at this link.

Week 8 learning outcomes

By the end of this week, you should be able to:

- recognise when AR or MA processes might not be adequate and explore ARMA processes as an alternative.
- fit ARMA processes to model short-term correlation.
- fit ARIMA processes to remove trend/seasonal patterns and model short-term correlation.
- forecast future values of a time series using linear and non-linear regression methods.
- forecast future values of a time series using exponential smoothing.
- forecast future values of a time series using an AR, MA or ARMA model.

Answers to tasks

Answer to Task 1. The characteristic polynomials are:

- $\phi(B) = 1 - 5B$ and $\theta(B) = 1 - 0.2B$.
- $\phi(B) = 1 - B + B^2$ and $\theta(B) = 1 + 0.1B$.

For the first process the characteristic equation for the AR part is $\phi(B) = 1 - 5B = 0$, which has a single root $B = 0.2 < 1$. Therefore the process is not stationary. The characteristic equation for the MA part is $\theta(B) = 1 - 0.2B = 0$, which has a single root $B = 5 > 1$. Therefore the process is invertible.

For the second process the characteristic equation for the AR part is $\phi(B) = 1 - B + B^2 = 0$, which has roots $B = \frac{1 \pm \sqrt{-3}}{2}$ with modulus equal to 1. Therefore the process is not stationary. The characteristic equation for the MA part is $\theta(B) = 1 + 0.1B = 0$, which has a single root $B = -10$ with modulus greater than 1. Therefore the process is invertible.

Answer to Task 2. The ACFs and PACFs do not look like either an AR(p) process or an MA(q) process, which makes identification difficult. A good approach in this case is to fit a simple ARMA(p, q) model, *e.g.* ARMA(1,1), and see if that removes the correlation. In fact, both time series processes X and Y were generated from different ARMA(1,1) models.

Answer to Task 3. This is an ARIMA(1,1,1) process.

In terms of X_t the model is given by

$$\begin{aligned}
(1 - B)(1 - 0.2B)X_t &= (1 - 0.5B)Z_t \\
(1 - 1.2B + 0.2B^2)X_t &= Z_t - 0.5Z_{t-1} \\
X_t - 1.2X_{t-1} + 0.2X_{t-2} &= Z_t - 0.5Z_{t-1} \\
X_t &= 1.2X_{t-1} - 0.2X_{t-2} - 0.5Z_{t-1} + Z_t
\end{aligned}$$

This process is not stationary, because the characterisitic equation for the AR(p) part has a root equal to one (as it is differenced once). Therefore it cannot be stationary.

Answer to Task 4. In backshift notation the model is given by

$$(1 - B)^2 X_t = (1 + \lambda B)Z_t.$$

In terms of X_t this model is given by

$$\begin{aligned}
(1 - B)(1 - B)X_t = (1 + \lambda B)Z_t & \\
(1 - 2B + B^2)X_t &= +\lambda Z_{t-1} + Z_t \\
X_t - 2X_{t-1} + X_{t-2} &= +\lambda Z_{t-1} + Z_t \\
X_t &= 2X_{t-1} - X_{t-2} + \lambda Z_{t-1} + Z_t
\end{aligned}$$

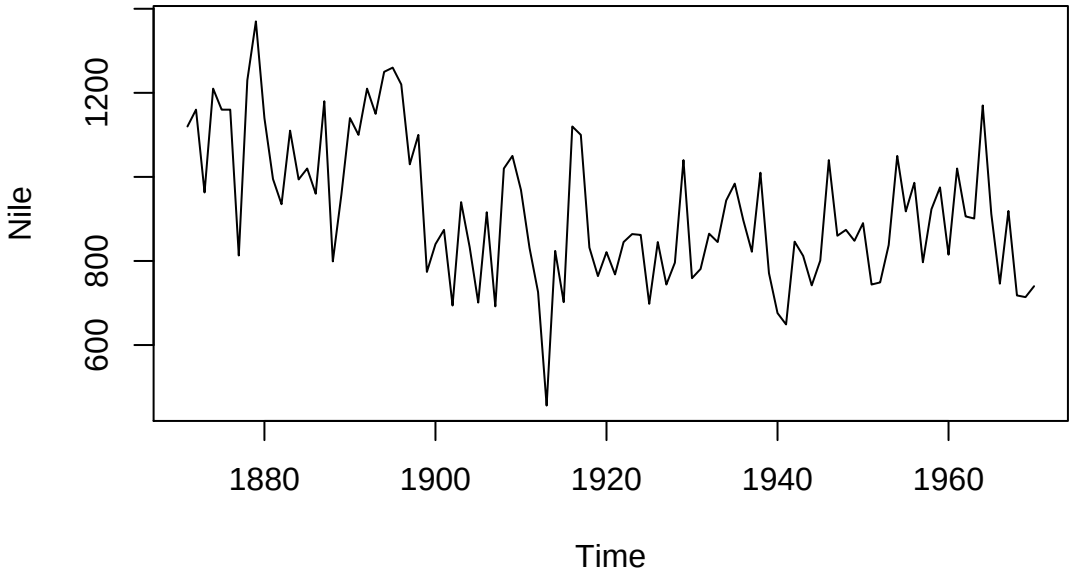
Answer to Task 5. ARIMA$(1, 0, 0)$: We cannot tell as it has an AR(1) component, so its stationarity will depend on the value of the lag one autocorrelation function.

ARIMA$(0, 0, 2)$: This is a purely a moving average process, and is therefore stationary.

ARIMA$(0, 1, 0)$: This has been differenced and is therefore not stationary.

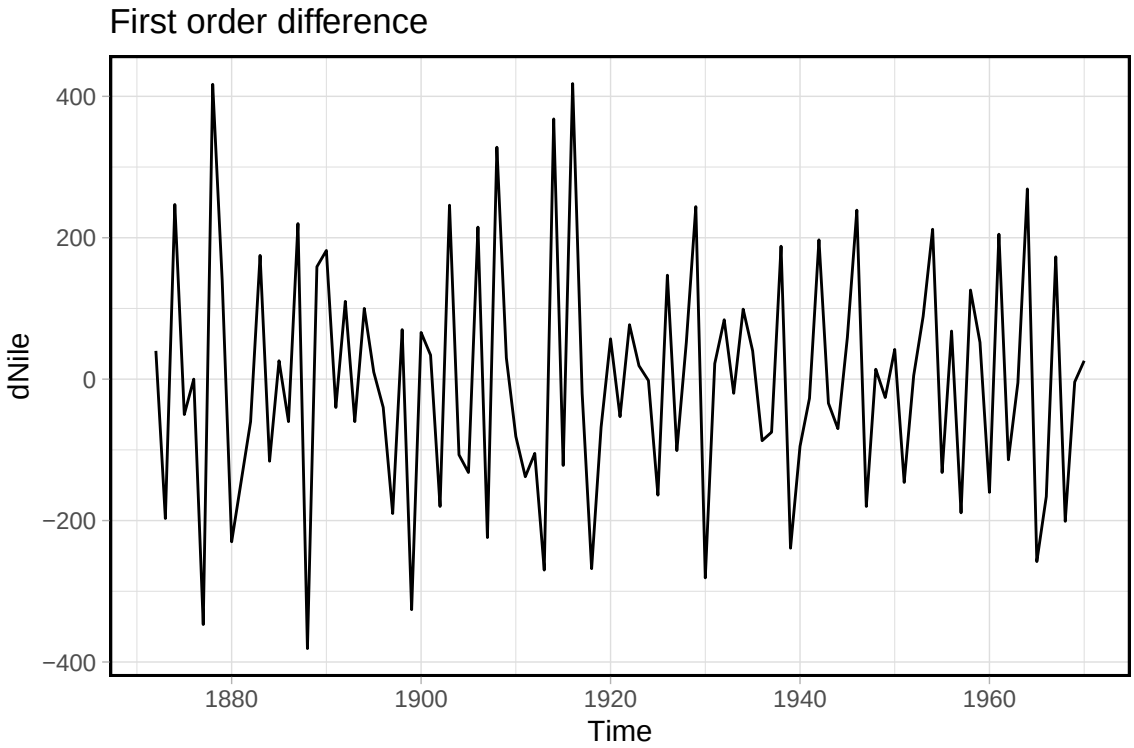
Answer to Task 6. Start by plotting the data:

```
plot(Nile, main="")
```



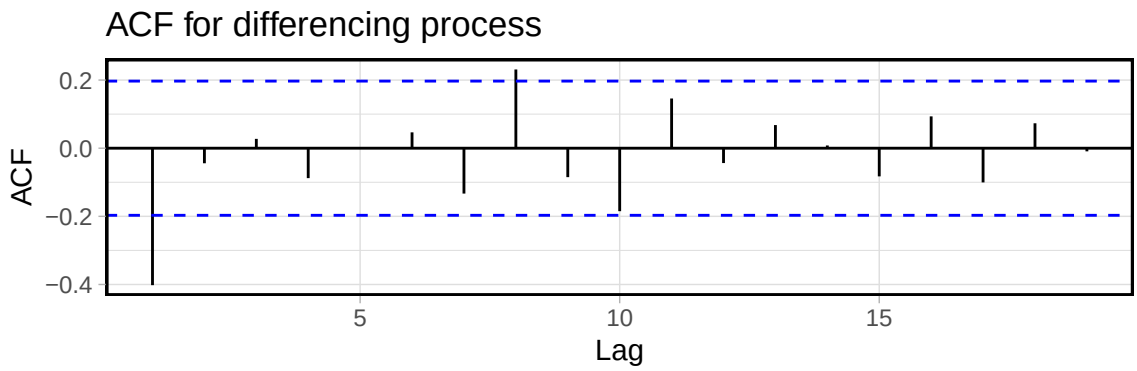
There is a decreasing trend in the series, so we difference once to see if it can be removed.

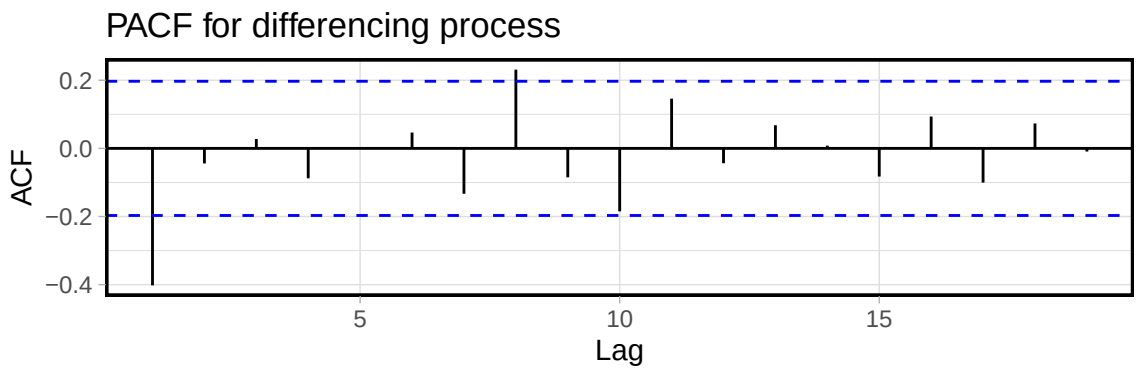
```
dNile <- diff(Nile)
autoplot(dNile, main="First order difference")
```



The differenced series appears stationary. Now look at the ACF and PACF plots to choose a suitable model to remove the short-term correlation:

```
diffacf <- autoplot(acf(dNile, plot = FALSE),  main="ACF for differencing process")
diffpacf <- autoplot(acf(dNile, plot = FALSE),  main="PACF for differencing process")
grid.arrange(diffacf, diffpacf, nrow=2)
```





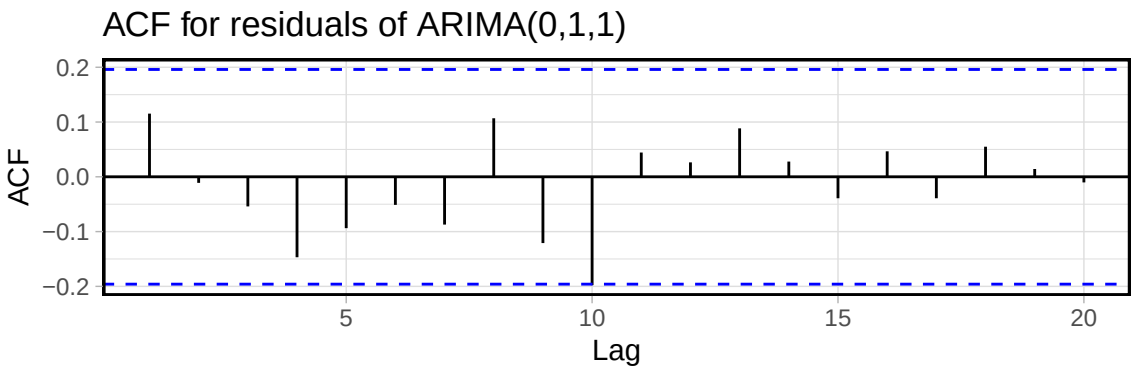
A large ACF at lag 1 and a PACF that tails off to zero suggest an MA(1) model as one possibility for this series, which would suggest an ARIMA(0,1,1) for the original `Nile` data.

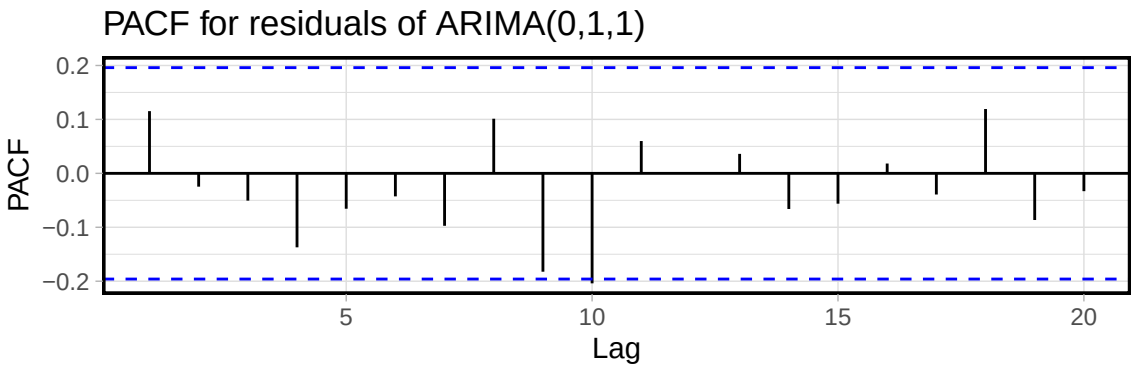
```
fitNile <- arima(Nile, order=c(0,1,1))
fitNile
```

```

Call:
arima(x = Nile, order = c(0, 1, 1))

Coefficients:
          ma1
      -0.7329
s.e.   0.1143

sigma^2 estimated as 20600:  log likelihood = -632.55,  aic = 1269.09
```

Finally check the residual series for any remaining short-term correlation:

```
resacf <- autoplot(acf(fitNile$residuals, plot = FALSE),
                   main="ACF for residuals of ARIMA(0,1,1)")
respacf <- autoplot(pacf(fitNile$residuals, plot = FALSE),
                    main="PACF for residuals of ARIMA(0,1,1)")
grid.arrange(resacf, respacf, nrow=2)
```

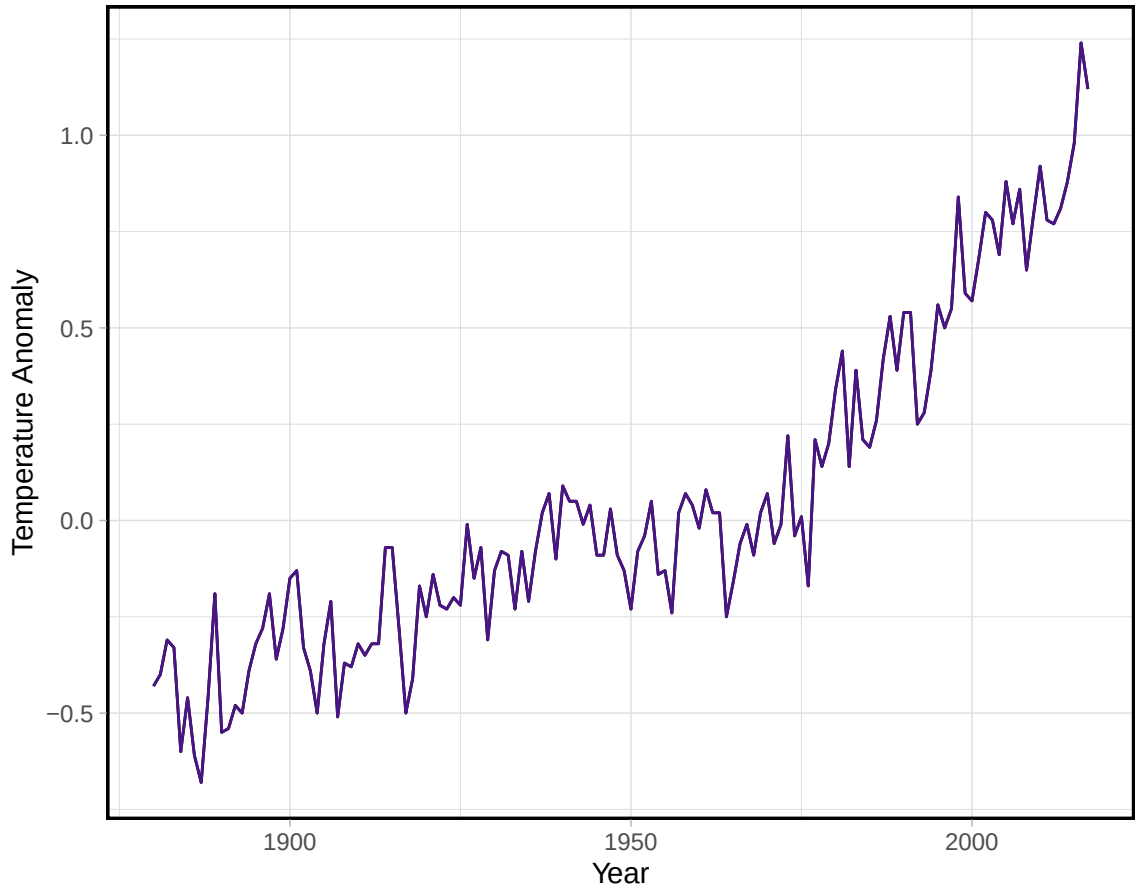




It looks like that the short-term correlation has been removed.

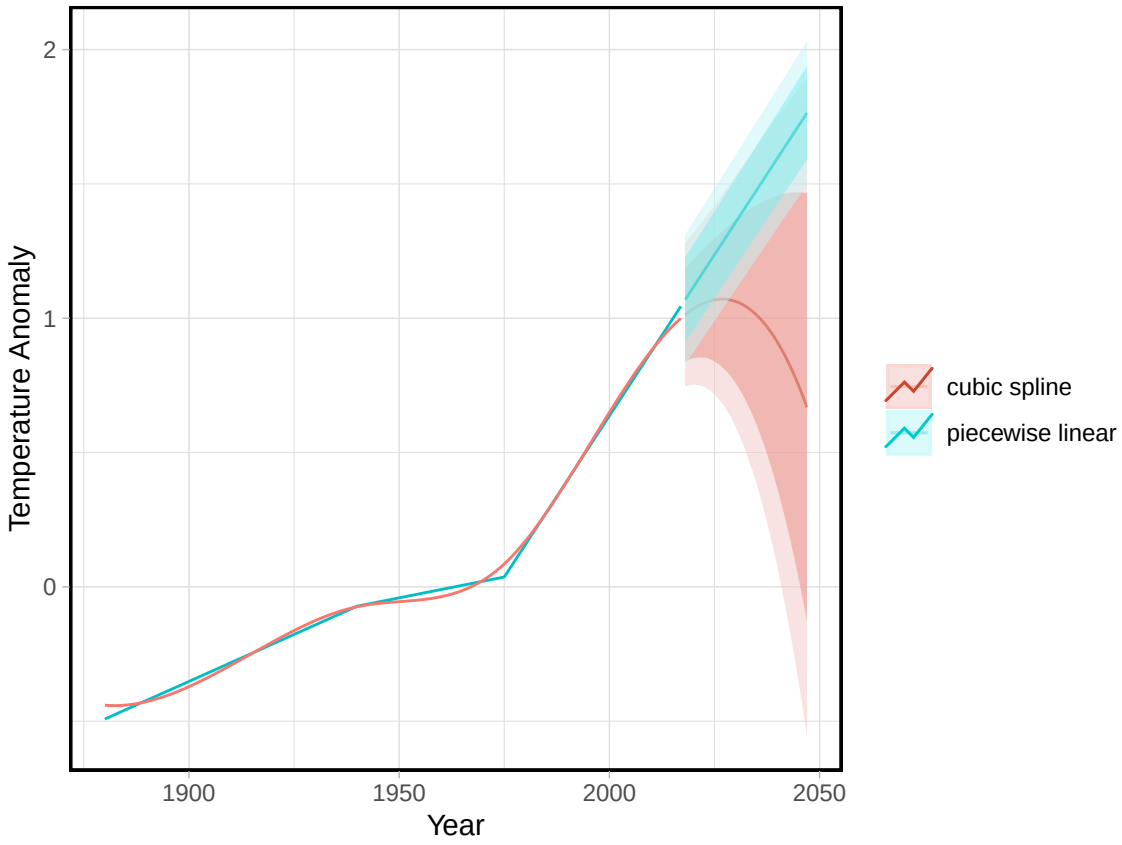
Answer to Task 7. Read in the data and plot the time series and its correlogram:

```
temp <- read.csv(url("http://www.stats.gla.ac.uk/~tereza/rp/globaltemp.csv"))

p <- autoplot(ts(temp$NoSmoothing, start=1880), main="", xlab="Year",
              ylab="Temperature Anomaly") + geom_line(colour="#4a1486")
p
```



There is an obvious trend. No changes in the variability is observed over time. Linear and cubic regression splines with knots at 1940 and 1975 give the following forecasts.

```
temp <- read.csv(url("http://www.stats.gla.ac.uk/~tereza/rp/globaltemp.csv"))
temp.ts <- ts(temp$NoSmoothing, start=1880)
Year <- temp$Year


Y.break1 <- 1940
Y.break2 <- 1975
Yb1 <- ts(pmax(0,temp$Year-Y.break1),start=1880)
Yb2 <- ts(pmax(0,temp$Year-Y.break2),start=1880)

temp.fit.lin <- tslm(temp.ts~ Year+Yb1+Yb2)
temp.fit.cub <- tslm(temp.ts~Year+I(Year^2)+I(Year^3)+I(Yb1^3)+I(Yb2^3))

# Forecasting
Year.new <- Year[length(Year)]+seq(30)
Yb1.new <- Yb1[length(Yb1)]+seq(30)
Yb2.new <- Yb2[length(Yb2)]+seq(30)
temp.newdata <- as.data.frame(cbind(Year=Year.new,Yb1=Yb1.new,Yb2=Yb2.new))
temp.fcasts.lin <- forecast(temp.fit.lin,newdata=temp.newdata)
temp.fcasts.cub <- forecast(temp.fit.cub,newdata=temp.newdata)

# Plotting
ggplot(temp)+
  autolayer(fitted(temp.fit.lin), series = "piecewise linear")+
  autolayer(fitted(temp.fit.cub), series = "cubic spline")+
  autolayer(temp.fcasts.cub, series = "cubic spline",alpha=0.5)+
  autolayer(temp.fcasts.lin, series = "piecewise linear",alpha=0.5)+
  ylab("Temperature Anomaly") + xlab("Year") +
  guides(colour=guide_legend(title=""))
```



The piecewise linear trend provides more reasonable forecasts. Note the narrower prediction intervals for the linear regression spline.

Answer to Task 8. The forecasts are given by

$$\hat{x}_n(1) = \alpha x_n = 0.1 \times 20 = 2$$

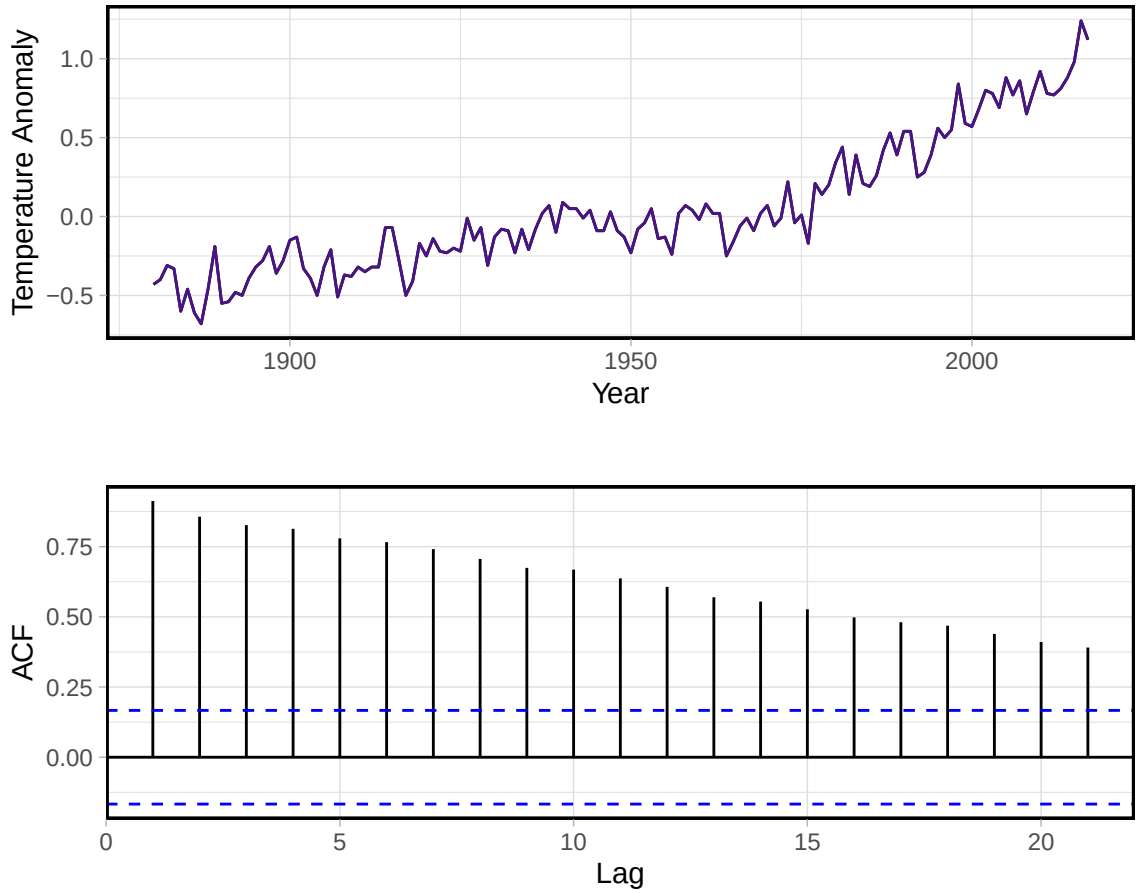
$$\hat{x}_n(2) = \alpha^2 x_n = 0.1^2 \times 20 = 0.2$$

and the error variances are

$$\mathrm{Var}(e_n(1)) = \hat{\sigma}_z^2 = 1 \qquad \text{and} \qquad \mathrm{Var}(e_n(2)) = \hat{\sigma}_z^2(1+\alpha^2) = 1.01$$

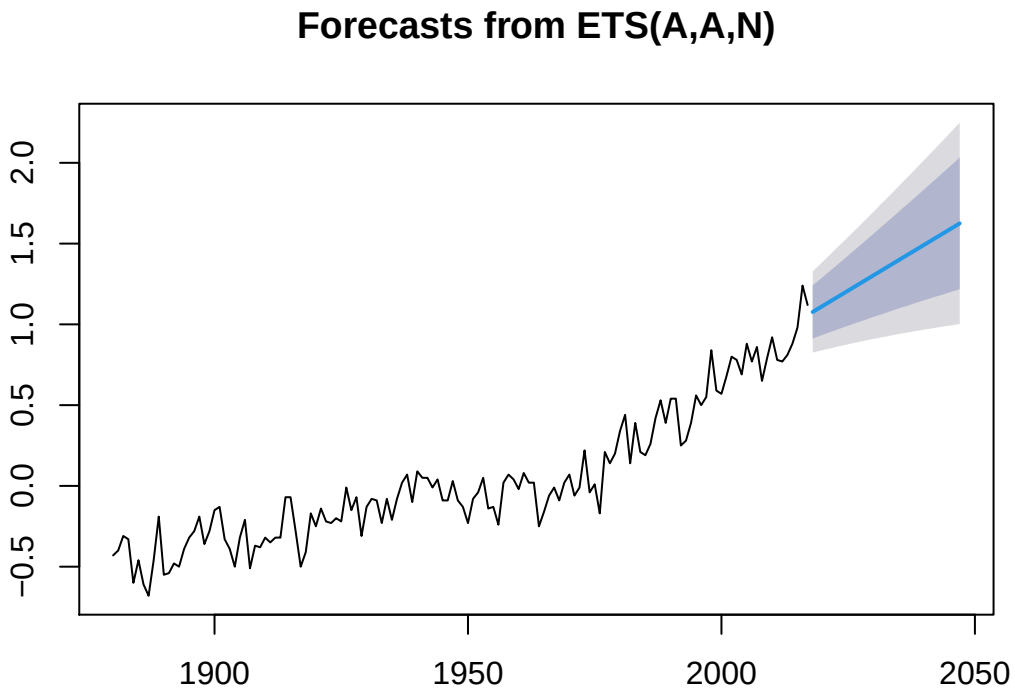
Answer to Task 9. Read in the data and plot the time series and its correlogram:

```
temp <- read.csv(url("http://www.stats.gla.ac.uk/~tereza/rp/globaltemp.csv"))

p <- autoplot(ts(temp$NoSmoothing, start=1880), main="", xlab="Year",
              ylab="Temperature Anomaly") + geom_line(colour="#4a1486")
acfp <- autoplot(acf(temp$NoSmoothing, plot = FALSE), main="",ylab="ACF")

grid.arrange(p, acfp, nrow=2)
```



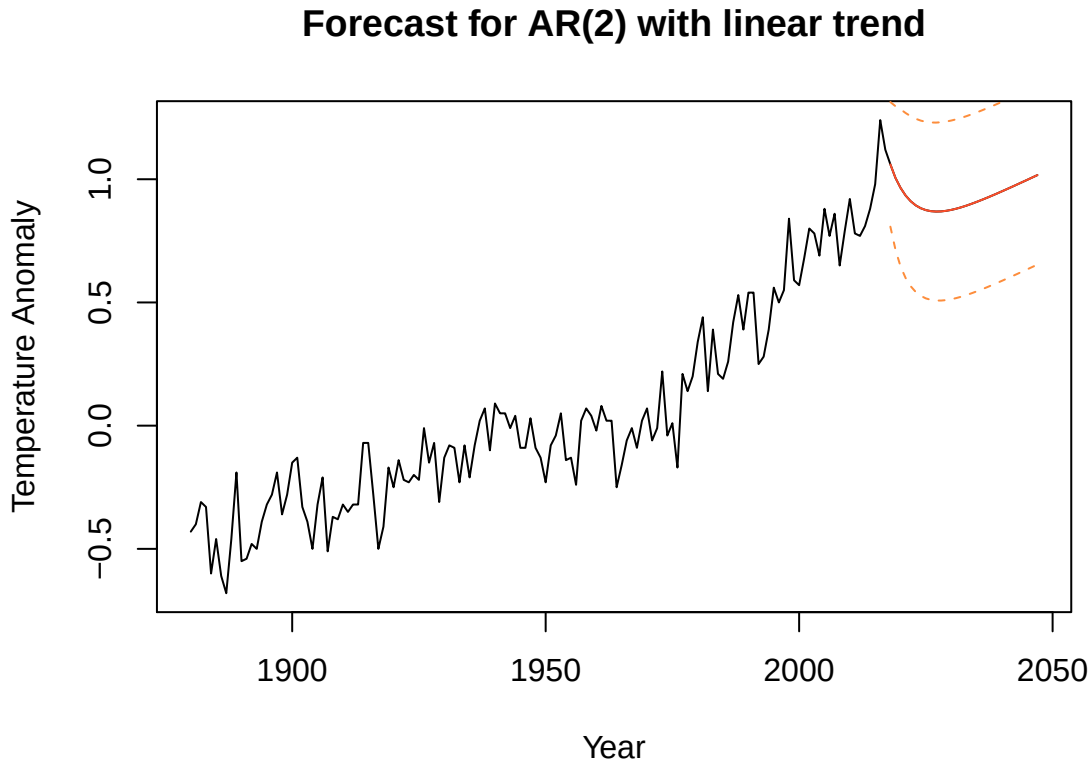
There is an obvious trend. Automatic exponential smoothing gives the following forecast.

```
temp <- read.csv(url("http://www.stats.gla.ac.uk/~tereza/rp/globaltemp.csv"))

ets.fit <- ets(ts(temp$NoSmoothing, start=1880))

plot(forecast(ets.fit, 30))
```


Forecasts from ETS(A,A,N)
−0.5
0.0
0.5
1.0
1.5
2.0
1900
1950
2000
2050

Try an AR(2) model with a linear trend:

```
year <- 1880:2017
year.predict <- 2018:2047
model.ar2 <- arima(temp$NoSmoothing, order=c(2,0,0), xreg=year, include.mean=TRUE)

# Prediction
predict.ar2 <- predict(model.ar2, n.ahead = 30, newxreg = year.predict, se.fit = TRUE)
ar2.LCI <- predict.ar2$pred - 1.96*predict.ar2$se
ar2.UCI <- predict.ar2$pred + 1.96*predict.ar2$se

# Plotting
plot(1880:2047, c(temp$NoSmoothing, predict.ar2$pred), type="l", xlab="Year",
     ylab="Temperature Anomaly", main="Forecast for AR(2) with linear trend")
lines(2018:2047, predict.ar2$pred, col="#fc4e2a")
lines(2018:2047, ar2.LCI, lty=2, col="#fd8d3c")
lines(2018:2047, ar2.UCI, lty=2, col="#fd8d3c")
```

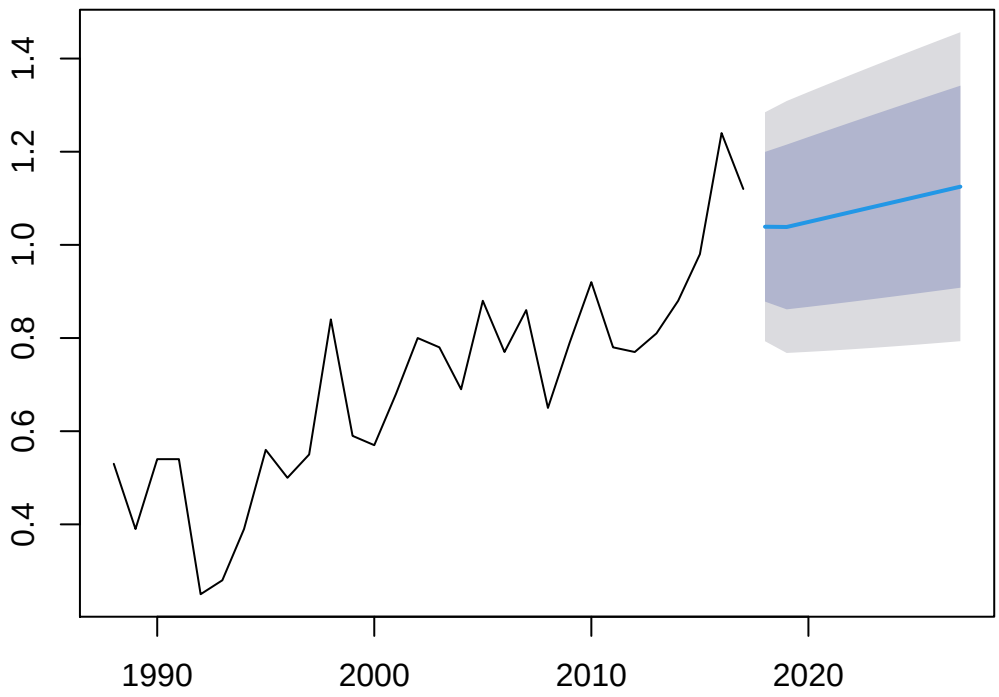


Finally an automated ARIMA forecast:

```
arima.fit <- auto.arima(ts(temp$NoSmoothing, start=1880))
arima.fit
```

```
Series: ts(temp$NoSmoothing, start = 1880)
ARIMA(0,1,2) with drift

Coefficients:
          ma1      ma2   drift
      -0.5404  -0.1834  0.0108
s.e.   0.0796   0.0753  0.0030

sigma^2 estimated as 0.01572:  log likelihood=91.27
AIC=-174.54   AICc=-174.24   BIC=-162.86
```

```
plot(forecast(arima.fit), 30)
```



This one is of the form

$$(1-\phi B)(1-B)(X_t-\beta t) = (1+\lambda B)Z_t$$

where the drift estimates the slope β. It gives the same results as

```
model.arima111 <- arima(temp$NoSmoothing, order=c(1,1,1), xreg=year, include.mean=TRUE)
model.arima111
```

```
Call:
arima(x = temp$NoSmoothing, order = c(1, 1, 1), xreg = year, include.mean = TRUE)

Coefficients:
         ar1      ma1    year
      0.2615  -0.8007  0.0108
s.e.  0.1095   0.0630  0.0029

sigma^2 estimated as 0.01539:  log likelihood = 91.21,  aic = -174.41
```

(For more details on including constants in ARIMA models please see this blog post.)

All three forecasts show an in increasing trend but are otherwise not very similar to each other. One thing they do have in common is the wide uncertainty intervals.