STAT 720 sp 2019 Lec 09

Karl Gregory

3/26/2019

This lecture will make use of the `tscourse` package, which is installed with the following R code:

```
library(devtools)
devtools::install_github("gregorkb/tscourse")
library(tscourse)
```

ARIMA models

ARIMA stands for "autoregressive-integrated moving average". The ARIMA model is a model for nonstationary time series which can be made into a stationary ARMA process by taking first-order differences a finite number of times.

An ARIMA(p, d, q) process is a process which, after differencing d times, becomes a causal ARMA(p, q) process. That is, we call $\{X_t, t \in \mathbb{Z}\}$ an ARIMA(p, d, q) process if the process $Y_t = (1 - B)^d X_t$ is a causal ARMA(p, q) process.

Recall that an ARMA(p, q) process $\{X_t, t \in \mathbb{Z}\}$ with the representation

$$\phi(B)X_t = \theta(B)Z_t, \quad t \in \mathbb{Z},$$

where $\{Z_t, t \in \mathbb{Z}\}$ is WN$(0, \sigma^2)$, is stationary if and only if the $\phi(u) \neq 0$ for all $|u| = 1$. Now, if $\{X_t, t \in \mathbb{Z}\}$ is an ARIMA(p, d, q) process, it has the representation

$$\phi(B)(1 - B)^d X_t = \phi^*(B)X_t = \theta(B)Z_t, \quad t \in \mathbb{Z},$$

where $\{Z_t, t \in \mathbb{Z}\}$ is WN$(0, \sigma^2)$ and $\phi^*(\cdot)$ is the polynomial given by $\phi^*(u) = \phi(u)(1 - u)^d$. Note that the polynomial $\phi^*(\cdot)$ has a root at $u = 1$ with multiplicity d; an ARIMA(p, d, q) process is therefore nonstationary for all $d > 0$.

Example of ARIMA$(1, 1, 0)$ process

For some $|\phi| < 1$, let $\{X_t, t \in \mathbb{Z}\}$ be the process satisfying

$$(1 - \phi B)(1 - B)X_t = Z_t, \quad t \in \mathbb{Z},$$

where $\{Z_t, t \in \mathbb{Z}\}$ is WN$(0, \sigma^2)$. This is an ARIMA$(1, 1, 0)$ process because the time series $Y_t = (1 - B)X_t$ is a causal ARMA$(1, 0)$, where causality comes from the fact that $|\phi| < 1$.

Generating data from an ARIMA$(p, 1, q)$ process

Consider how we might generate data from an ARIMA$(p, 1, q)$ process. Let $\{Y_t, t \in \mathbb{Z}\}$ be a causal ARMA(p, q) process and let $\{X_t, t \in \mathbb{Z}\}$ be the time series such that $Y_t = (1 - B)X_t$ for all $t \in \mathbb{Z}$.

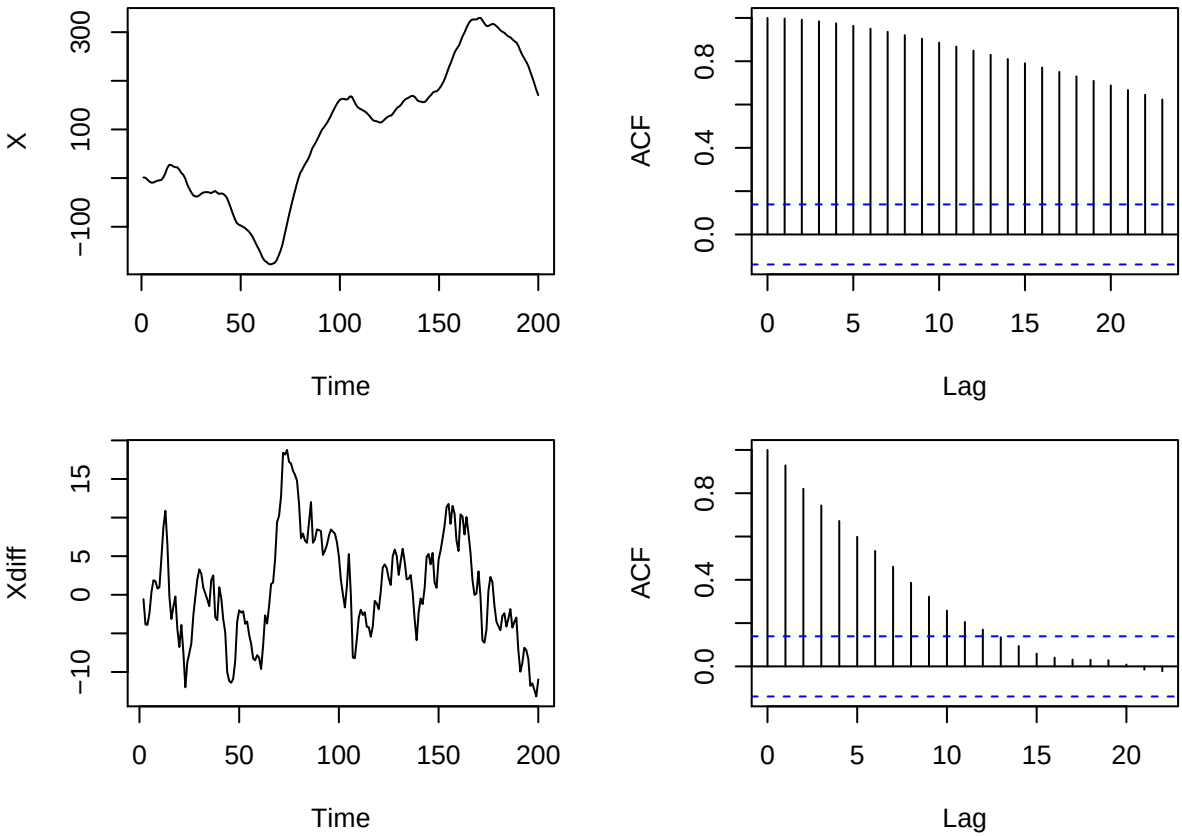
Then for each $t > 0$ we may write

$$
\begin{aligned}
X_t &= X_{t-1} + Y_t \\
&= (X_{t-2} + Y_{t-1}) + Y_t \\
&= ((X_{t-3} + Y_{t-2}) + Y_{t-1}) + Y_t \\
&\vdots \\
&= X_0 + \sum_{j=1}^{t} Y_j.
\end{aligned}
$$

Thus in order to generate a length-n realization $X_1, \ldots, X_n$ of the ARIMA$(p, 1, q)$ time series $\{X_t, t \in \mathbb{Z}\}$, we may begin by setting $X_0 = 0$, say, and then generating $Y_1, \ldots, Y_n$ from the causal ARMA(p, q) process $\{Y_t, t \in \mathbb{Z}\}$, which we already know how to do. Then we construct $X_1, \ldots, X_n$ according to $X_t = X_0 + \sum_{j=1}^{t} Y_j$ for $t = 1, \ldots, n$.

The following R code performs an example of this.

```r
# generate Y from causal ARMA(p,q) process
phi <- c(.5,.3)
theta <- c(.8)
sigma <- 2
n <- 200
Y <- get.ARMA.data(phi,theta,sigma,n)

# construct X from Y
X <- as.ts(cumsum(Y))

par(mfrow=c(2,2),mar=c(4.1, 4.1, 1.1, 2.1))

# see non-stationarity in plot
plot(X)

# see slowly decaying acf
acf(X,main="")

# take 1st-order differences and plot
Xdiff <- diff(X)
plot(Xdiff)

# see rapidly decaying acf
acf(Xdiff,main="")
```



In order to fit an ARIMA model to real data, we begin by finding the appropriate number of times to difference the data. In most cases we will find that differencing once, $d = 1$, is sufficient, sometimes twice, $d = 2$. Seldom do we take first order differences more than twice in practice. After determining d, we must make choices of p and q. This is the focus of the next section.

ARMA model selection with AIC and BIC

Akaike's information criterion (AIC) is a commonly-used model selection tool. It is often used to determine the appropriate size of a model in terms of how many parameters should be included. In the context of regression, it can be used to select the number of covariates which should be included in the model; in the context of fitting ARMA models, it can be used as a way to select p and q. The AIC is based on the log-likelihood function. Given a log-likelihood function $\ell(\theta)$, where θ is a vector of parameters, the AIC criterion is defined as

$$\text{AIC} = -2\ell(\hat{\theta}) + 2\dim\hat{\theta},$$

where $\hat{\theta}$ is the maximum likelihood estimator of θ and $\dim\hat{\theta}$ is the number of estimated parameters in $\hat{\theta}$. One chooses the model with the smallest AIC value. Thus the term $2\dim\hat{\theta}$ acts as a penalty for including more parameters in the model.

Since the AIC is a likelihood-based criterion, we must make a choice of distribution for our data. It is conventional to use the Gaussian likelihood to choose p and q according to AIC even if the time series is not Gaussian. We compute the AIC for an ARMA(p, q) model as follows:

Let $X_1, \ldots, X_n$ be some observed data and suppose we wish to estimate the parameters of an ARMA$(\hat{p}, \hat{q})$ model based on $X_1, \ldots, X_n$, where $\hat{p}$ and $\hat{q}$ are candidate values of p and q. Then the Gaussian likelihood

based on $X_1, \ldots, X_n$ is given by

$$L(\boldsymbol{\phi}, \boldsymbol{\theta}, \sigma^2; X_1, \ldots, X_n) = (2\pi)^{-n/2} |v_0 \cdots v_{n-1}|^{-1/2} \exp\left[-\frac{1}{2} \sum_{j=1}^{n} (X_j - \hat{X}_j)^2 / v_{j-1} \right],$$

and the log-likelihood is given by

$$\ell(\boldsymbol{\phi}, \boldsymbol{\theta}, \sigma^2; X_1, \ldots, X_n) = -\frac{n}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^{n} \log v_{j-1} - \frac{1}{2} \sum_{j=1}^{n} (X_j - \hat{X}_j)^2 / v_{j-1},$$

where $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_{\hat{p}})^T$ and $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_{\hat{q}})^T$ and $\hat{X}_1, \ldots, \hat{X}_n$ are the one-step-ahead predictions of $X_1, \ldots, X_n$ and $v_0, \ldots, v_{n-1}$ are the associated MSEPs based on the ARMA$(\hat{p}, \hat{q})$ model. Let $\hat{\boldsymbol{\phi}} = (\hat{\phi}_1, \ldots, \hat{\phi}_{\hat{p}})^T$ and $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_{\hat{q}})^T$ and $\hat{\sigma}^2$ be the maximum likelihood estimators of $\boldsymbol{\phi}$, $\boldsymbol{\theta}$, and σ^2. Then the AIC is given by

$$\text{AIC} = -2\ell(\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\theta}}, \hat{\sigma}^2; X_1, \ldots, X_n) + 2(\hat{p} + \hat{q} + 1),$$

where $\hat{p} + \hat{q} + 1$ is the number of parameters estimated, including σ^2.

In order to select p and q, we may fit ARMA models for many combinations of p and q, for example for all $(p, q) \in \{(i, j), 0 \leq i, j \leq 8\}$ and compute the AIC for each one. Then we may select p and q pair for which the AIC is the smallest. We seldom choose p or q to be much greater than 3, so we do not need to consider a very large number of candidate models.

After selecting a model via AIC, we should check to see if the residuals behave like white noise using, for example, the Ljung-Box test.

The following R code illustrates how we can choose p and q based on the AIC. The code also computes the BIC criterion, which is explained below.

```
# generate some ARMA data
phi <- c(.5,.3)
theta <- c(.1)
sigma <- 2
n <- 100
X <- get.ARMA.data(phi,theta,sigma,n)

# true values of p and q
p <- length(phi)
q <- length(theta)

p.hat <- c(0:5)
q.hat <- c(0:5)
AIC <- AIC.arima <- matrix(NA,length(p.hat),length(q.hat))
BIC <- BIC.arima <- matrix(NA,length(p.hat),length(q.hat))
for( j in 1:length(p.hat) )
    for( k in 1:length(q.hat) )
    {

        # get AIC from arima()
        arima.out <- arima(X-mean(X),order=c(p.hat[j],0,q.hat[k]),include.mean=FALSE)
        AIC.arima[j,k] <- arima.out$aic
        BIC.arima[j,k] <- BIC(arima.out)

        # compute AIC using ARMA.hstep() function from ts course package
        if( (p.hat[j] == 0) & (q.hat[k] == 0) )
```

```
        {

            phi.hat <- NULL
            theta.hat <- NULL

        } else if (p.hat[j] == 0 ){

            phi.hat <- NULL
            theta.hat <- arima.out$coef[1:q.hat[k]]

        } else if (q.hat[k] == 0){

            phi.hat <- arima.out$coef[1:p.hat[j]]
            theta.hat <- NULL

        } else {

            phi.hat <- arima.out$coef[1:p.hat[j]]
            theta.hat <-arima.out$coef[-c(1:p.hat[j])]
        }

        sigma.hat <- sqrt(arima.out$sigma2)

        ARMA.hstep.out <- ARMA.hstep(X,1,phi.hat,theta.hat,sigma.hat)
        AIC[j,k] <- ARMA.hstep.out$aic
        BIC[j,k] <- ARMA.hstep.out$bic

    }
```

```
## Warning in arima(X - mean(X), order = c(p.hat[j], 0, q.hat[k]),
## include.mean = FALSE): possible convergence problem: optim gave code = 1

## Warning in arima(X - mean(X), order = c(p.hat[j], 0, q.hat[k]),
## include.mean = FALSE): possible convergence problem: optim gave code = 1

## Warning in arima(X - mean(X), order = c(p.hat[j], 0, q.hat[k]),
## include.mean = FALSE): possible convergence problem: optim gave code = 1

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced
```

```
# note that the AIC from ARMA.hstep matches the arima-computed AIC:
AIC
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,] 446.3320 414.2164 402.0005 401.2599 398.7279 398.9315
## [2,] 392.4403 391.2762 393.2744 395.2740 395.9152 397.6818
## [3,] 391.5695 393.2744 395.2760 397.0559 394.5386 396.3192
## [4,] 393.3850 395.2736 397.1388 395.4939 395.1917 397.4655
## [5,] 394.8729 396.6977 395.2065 397.1567 399.9291 398.3055
## [6,] 396.4836 394.6533 397.9083 399.1540 400.6036 399.4871
```

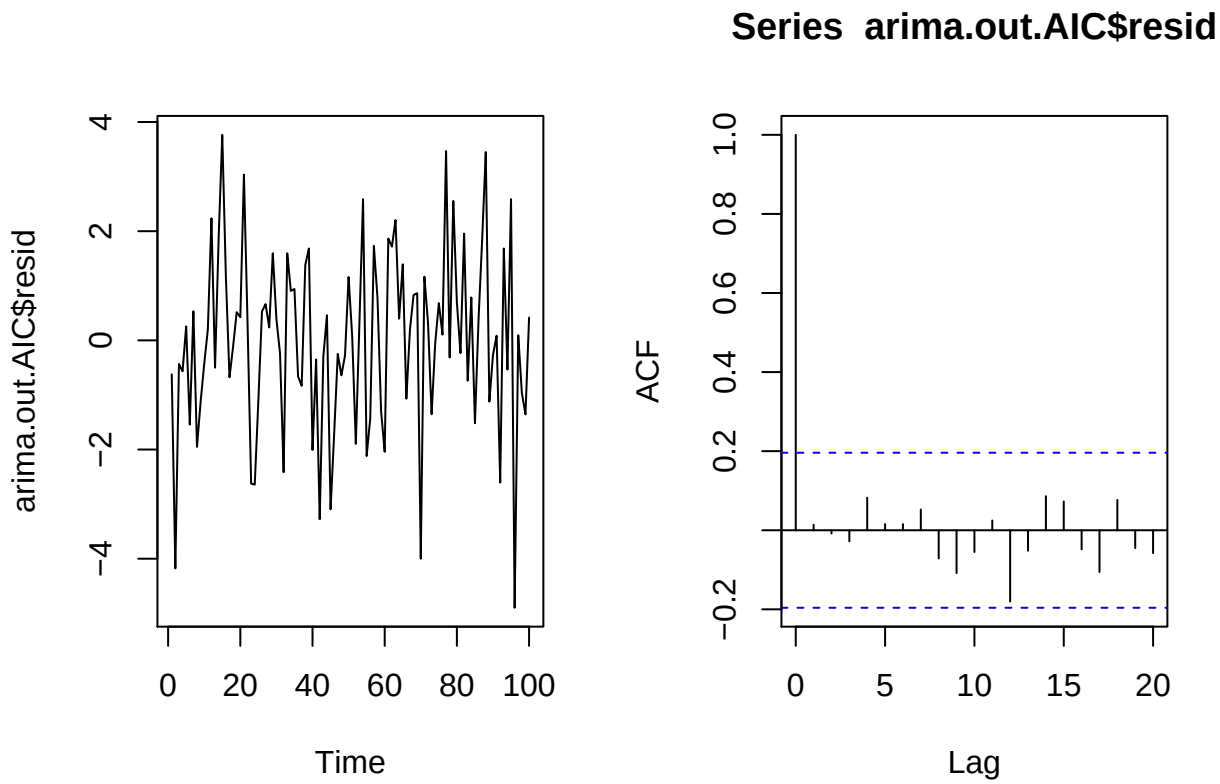
```
AIC.arima
```

```
##           [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,] 446.3320 414.2164 402.0005 401.2599 398.7279 398.9315
## [2,] 392.4403 391.2762 393.2744 395.2740 395.9152 397.6818
## [3,] 391.5695 393.2744 395.2760 397.0559 394.5386 396.3192
## [4,] 393.3850 395.2736 397.1388 395.4939 395.1917 397.2181
## [5,] 394.8729 396.6977 395.2065 397.1567 399.9291 398.3055
## [6,] 396.4836 394.6533 396.9855 399.1540 398.4237 399.4871
```

```
# identify model with lowest AIC
which.AIC <- which(AIC == min(AIC),arr.ind=TRUE)
p.AIC <- p.hat[which.AIC[1]]
q.AIC <- q.hat[which.AIC[2]]

c(p.AIC,q.AIC)
```

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

```
# fit model with lowest AIC:
arima.out.AIC <- arima(X-mean(X),order=c(p.AIC,0,q.AIC),include.mean=FALSE)

par(mfrow=c(1,2))
# look at residuals
plot(arima.out.AIC$resid)

# look at acf of residuals
acf(arima.out.AIC$resid)
```




```
# test whether residuals are white noise by Ljung-Box
WNtest.LB(arima.out.AIC$resid,k = p.AIC + q.AIC + 1,nparms = p.AIC + q.AIC)
```

```
## [1] 0.7395025
```

It may happen that the `arima()` function gives a warning about possible convergence issues. These can happen when trying to fit ARMA models which are unnecessarily large.

We may also consider the model chosen by the BIC criterion, which is defined in general by

$$\text{BIC} = -2\ell(\hat{\theta}) + \log(n) \dim \hat{\theta},$$

so that for choosing an ARMA model we consider

$$\text{BIC} = -2\ell(\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\theta}}, \hat{\sigma}^2; X_1, \ldots, X_n) + \log(n)(\hat{p} + \hat{q} + 1).$$

The AIC criterion is known to have a tendency to select larger-than-necessary models, but practitioners are for some reason very attached to it. The BIC generally chooses models with fewer parameters, as it more strongly penalizes the inclusion of additional parameters.

```
# note that the AIC from ARMA.hstep matches the arima-computed AIC:
BIC
```

```
##           [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,] 448.9372 419.4267 409.8160 411.6806 411.7537 414.5626
## [2,] 397.6506 399.0917 403.6951 408.2999 411.5463 415.9180
## [3,] 399.3850 403.6951 408.3019 412.6869 412.7748 417.1605
## [4,] 403.8056 408.2994 412.7698 413.7301 416.0330 420.9120
## [5,] 407.8987 412.3287 413.4427 417.9981 423.3756 424.3572
## [6,] 412.1146 412.8895 418.7496 422.6005 426.6553 428.1440
```

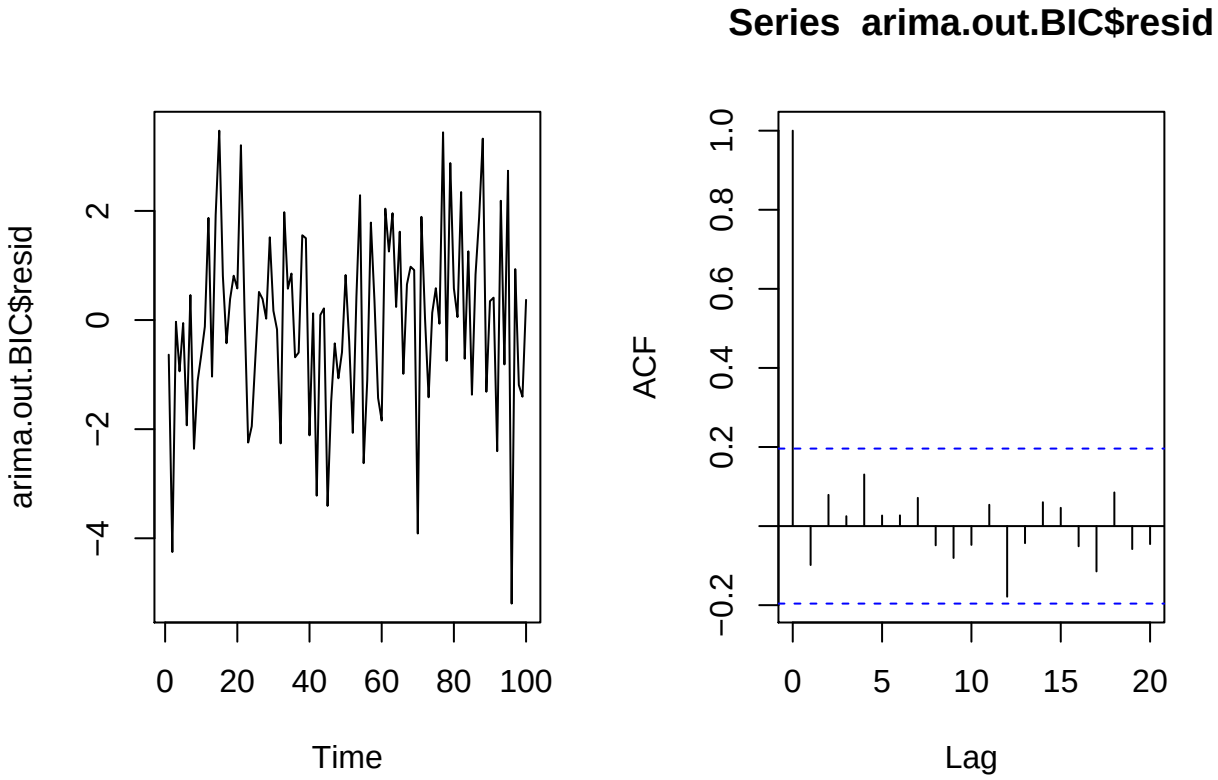
```
BIC.arima
```

```
##           [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,] 448.9372 419.4267 409.8160 411.6806 411.7537 414.5626
## [2,] 397.6506 399.0917 403.6951 408.2999 411.5463 415.9180
## [3,] 399.3850 403.6951 408.3019 412.6869 412.7748 417.1605
## [4,] 403.8056 408.2994 412.7698 413.7301 416.0330 420.6646
## [5,] 407.8987 412.3287 413.4427 417.9981 423.3756 424.3572
## [6,] 412.1146 412.8895 417.8269 422.6005 424.4754 428.1440
```

```
# identify model with lowest BIC
which.BIC <- which(BIC == min(BIC),arr.ind=TRUE)
p.BIC <- p.hat[which.BIC[1]]
q.BIC <- q.hat[which.BIC[2]]

c(p.BIC,q.BIC)
```

```
## [1] 1 0
```

```
# fit model with lowest BIC:
arima.out.BIC <- arima(X-mean(X),order=c(p.BIC,0,q.BIC),include.mean=FALSE)

par(mfrow=c(1,2))
# look at residuals
plot(arima.out.BIC$resid)

# look at acf of residuals
acf(arima.out.BIC$resid)
```




```
# test whether residuals are white noise by Ljung-Box
WNtest.LB(arima.out.BIC$resid, k = p.BIC + q.BIC + 1, nparms = p.BIC + q.BIC)
```

```
## [1] 0.1993972
```

Forecasting for ARIMA processes

If $\{X_t, t \in \mathbb{Z}\}$ is an ARIMA(p, d, q) process then the process given by $Y_t = (1 - B)^d X_t$ for $t \in \mathbb{Z}$ is an ARMA(p, q) process. We construct forecasts of X_{n+h}, $h \geq 1$ using forecasts of $Y_{n+1}, \ldots, Y_{n+h}$ in the following way.

We may write

$$X_t = Y_t - \sum_{j=1}^{d} \binom{d}{j} (-1)^j X_{t-j}, \quad t = 1, \ldots, n, \tag{1}$$

because

$$(1 - u)^d = \sum_{j=0}^{d} \binom{d}{j} (-u)^j (1)^{d-j} = \sum_{j=0}^{d} \binom{d}{j} (-1)^j u^j = 1 + \sum_{j=1}^{d} \binom{d}{j} (-1)^j u^j.$$

Therefore we have

$$(1 - B)^d X_t = X_t + \sum_{j=1}^{d} \binom{d}{j} (-1)^j X_{t-j},$$

from which we may write (1). Note that this representation of X_t for $t = 1, \ldots, n$ requires us to define $X_{1-d}, X_{2-d}, \ldots, X_0$. Given observed data $X_1, \ldots, X_{n^*}$, we solve this problem by re-indexing the data

according to the diagram below, in which $n = n^* - d$:

				Y_1	$\dots$	Y_n
X_{1-d}	X_{2-d}	$\dots$	X_0	X_1	$\dots$	X_n

Now, we wish to predict X_{n+h} by projecting it on the the space spanned by the random variables $X_{1-d}, \dots, X_n$. Recall that

$$P_{\overline{\text{sp}}\{X_{1-d},\dots,X_n\}} X_{n+h} = \sum_{j=1}^{n+d} a_j X_{n+1-j},$$

where $a_1, \dots, a_{n+d}$ are chosen to minimize

$$\mathbb{E}\left[X_{n+h} - \sum_{j=1}^{n+d} a_j X_{n+1-j}\right]^2.$$

Let $S_n = \overline{\text{sp}}\{X_{1-d}, \dots, X_n\}$. Then because of (1) we have $S_n = \overline{\text{sp}}\{X_{1-d}, \dots, X_0, Y_1, \dots, Y_n\}$, that is, we can express any of $X_1, \dots, X_n$ as linear combinations of the random variables $X_{1-d}, \dots, X_0, Y_1, \dots, Y_n$ because of equation (1), giving

$$\overline{\text{sp}}\{X_{1-d}, \dots, X_n\} = \overline{\text{sp}}\{X_{1-d}, \dots, X_0, Y_1, \dots, Y_n\}.$$

Define $S_0 = \overline{\text{sp}}\{X_{1-d}, \dots, X_0\}$ and assume

$$\overline{\text{sp}}\{X_{1-d}, \dots, X_0\} \perp \overline{\text{sp}}\{Y_1, \dots, Y_n\}, \tag{2}$$

that is, assume that the random variables $Y_1, \dots, Y_n$ are uncorrelated with the random variables $X_{1-d}, \dots, X_0$.

Now we consider the projection of Y_{n+h} onto $S_n = \overline{\text{sp}}\{X_{1-d}, \dots, X_0, Y_1, \dots, Y_n\}$. We have

$$P_{S_n} Y_{n+h} = \underbrace{P_{S_0} Y_{n+h}}_{=0 \text{ by } (2)} + P_{\overline{\text{sp}}\{Y_1,\dots,Y_n\}} Y_{n+h}. P_{\overline{\text{sp}}\{Y_1,\dots,Y_n\}} Y_{n+h}$$

Therefore, applying the projection P_{S_n} to both sides of (1) with $t = n + h$ gives

$$\begin{aligned} P_{S_n} X_{n+h} &= P_{S_n} Y_{n+h} + \sum_{j=1}^{d} \binom{d}{j} (-1)^j P_{S_n} X_{t-j} \\ &= \underbrace{P_{\overline{\text{sp}}\{Y_1,\dots,Y_n\}} Y_{n+h}}_{\text{ARMA prediction}} + \sum_{j=1}^{d} \binom{d}{j} (-1)^j P_{S_n} X_{t-j}, \end{aligned}$$

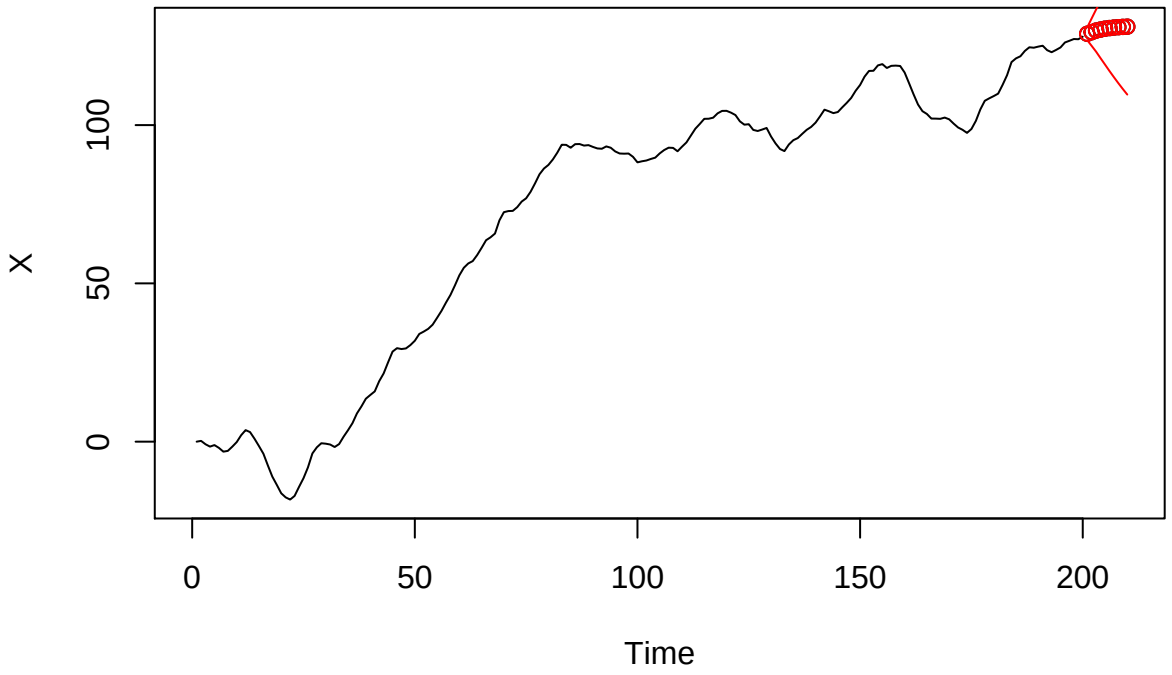
by which the h-step-ahead predictions can be computed recursively for $h = 1, 2, \dots$

The MSEPs for the h-step-ahead predictions are quite complicated. See Section 9.5 of B&D Theory.

The following R code demonstrates how to recursively compute h-step ahead forecasts for an ARIMA(p, d, q) model. It also shows how to get the predictions directly using the `arima()` function, which also returns the MSEPs, allowing, in the case of a Gaussian time series, the construction of prediction intervals.

```
# generate data from an ARIMA(p,d,q) model:
phi <- c(.5,.2)
theta <- c(.3)
sigma <- 2
n <- 200
```

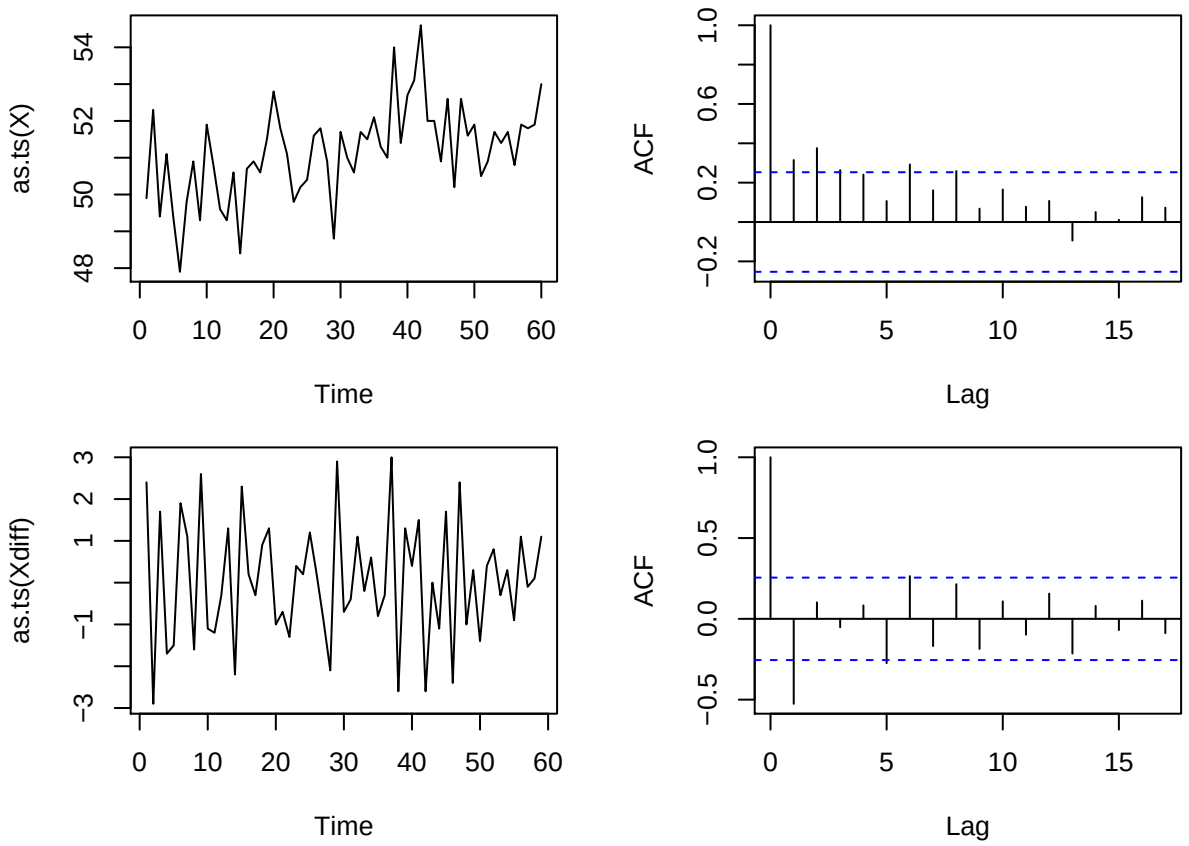
```r
p <- length(phi)
q <- length(theta)
d <- 1 # the code works for d > 1 as well

# this time use arima.sim() function to generate the ARIMA(p,d,q) data
# note that arima.sim() returns a realization of length n+d when n = n,
# so we put n = n-d to get a realization of length n.
X <- arima.sim(model=list(order=c(p,d,q), ar= phi, ma = theta),n-d)

# difference the series d times and fit ARMA(p,q) model
# on differenced series, assuming it has zero mean
Xdiff <- X
for(j in 1:d)
{
  Xdiff <- diff(Xdiff)
}
arima.out <- arima(Xdiff,order=c(p,0,q),include.mean=FALSE)

# recursively compute h-step-ahead predictions based on predictions
# of differenced series; get the ARMA predictions from predict.Arima()
h <- 10
Xdiff.hpred <- predict(arima.out,n.ahead=h)$pred
X.hpred <- numeric()
for(j in 1:h)
{

    X.hpred[j] <- Xdiff.hpred[j] - sum(choose(d,1:d)*(-1)^(1:d)*c(X,X.hpred)[n+j-1:d])

}

# plot predicted values
plot(X,xlim=c(0,n+h),ylim=range(X,X.hpred))
points(X.hpred~c((n+1):(n+h)))

# check these prediction values against the predicted values obtained the easy way,
# from predict.Arima() directly:
arima2.out <- arima(X,order=c(p,d,q))
predict.arima2.out <- predict(arima2.out,n.ahead=h)
X.hpred.arima <- predict.arima2.out$pred
points(X.hpred.arima~c((n+1):(n+h)),col="red")

# put prediction intervals around predictions using arima() std errors
X.hpred.se.arima <- predict.arima2.out$se
lines(X.hpred.arima + 1.96 * X.hpred.se.arima ~c((n+1):(n+h)),col="red")
lines(X.hpred.arima - 1.96 * X.hpred.se.arima ~c((n+1):(n+h)),col="red")
```



Note that the prediction intervals become very wide very quickly.

Data example

```
# import nhtemp data
data(nhtemp)
X <- as.numeric(nhtemp)
n <- length(X)

# take d=1 differences and plot series; looks stationary
Xdiff <- diff(X)
par(mfrow=c(2,2),mar=c(4.1, 4.1, 1.1, 2.1))
plot(as.ts(X))
acf(X,main="")
plot(as.ts(Xdiff))
acf(Xdiff,main="")
```



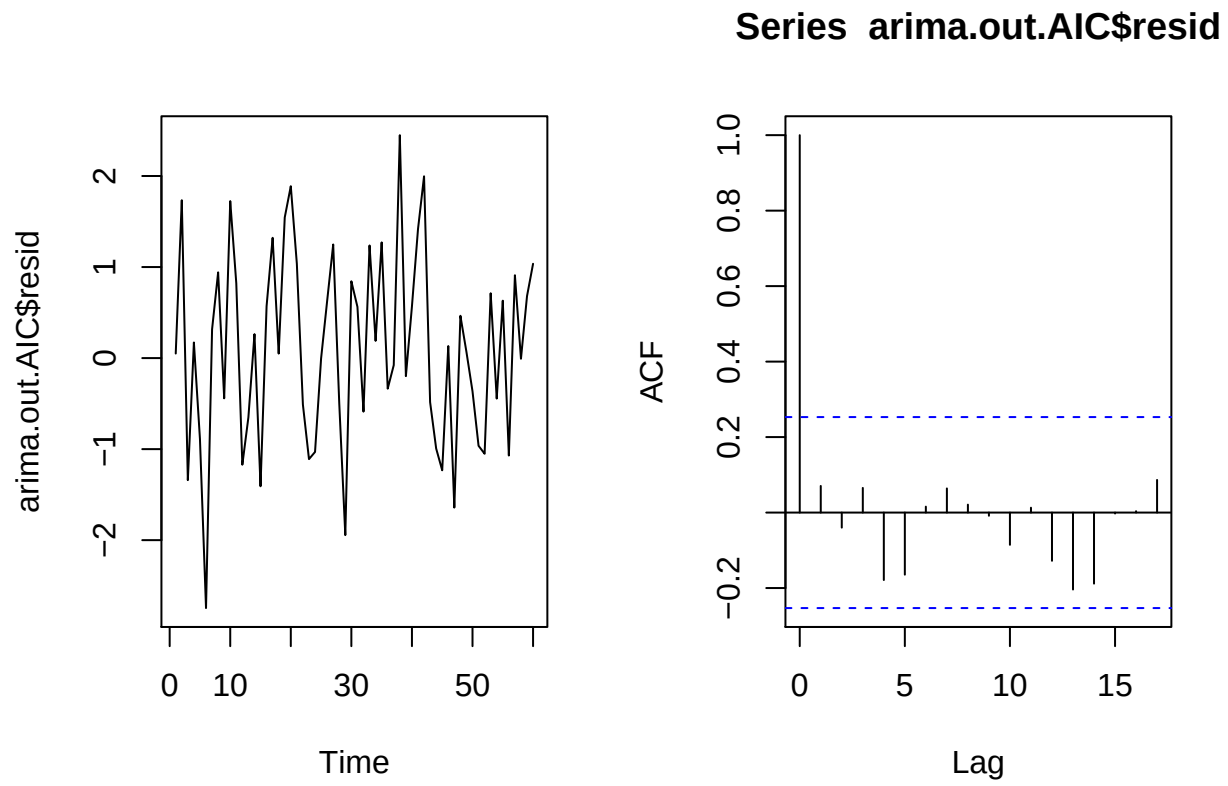
The differenced series appears to be a stationary time series. So we now try to find a suitable ARMA(p, q) model for the differenced series.

```
# consider all (p,q) pairs such that 0 <= p,j,<= 3
p.hat <- c(0:3)
q.hat <- c(0:3)
AIC <- BIC <- matrix(NA,length(p.hat),length(q.hat))

for( j in 1:length(p.hat) )
    for( k in 1:length(q.hat) )
    {

        # get AIC from arima()
      try({
        arima.out <- arima(Xdiff,order=c(p.hat[j],0,q.hat[k]),include.mean=FALSE)
        AIC[j,k] <- arima.out$aic
        BIC[j,k] <- BIC(arima.out)
      })
    }

# identify model with lowest AIC
which.AIC <- which(AIC == min(AIC),arr.ind=TRUE)
p.AIC <- p.hat[which.AIC[1]]
q.AIC <- q.hat[which.AIC[2]]

# fit model with lowest AIC:
arima.out.AIC <- arima(X,order=c(p.AIC,1,q.AIC))
```

```
arima.out.AIC
```

```
##
## Call:
## arima(x = X, order = c(p.AIC, 1, q.AIC))
##
## Coefficients:
##           ar1     ma1      ma2
##       -0.9997  0.2335  -0.7518
## s.e.   0.0022  0.1196   0.1180
##
## sigma^2 estimated as 1.169:  log likelihood = -89.56,  aic = 187.12
```

```
# look at residuals
par(mfrow=c(1,2))
plot(arima.out.AIC$resid)
acf(arima.out.AIC$resid)
```

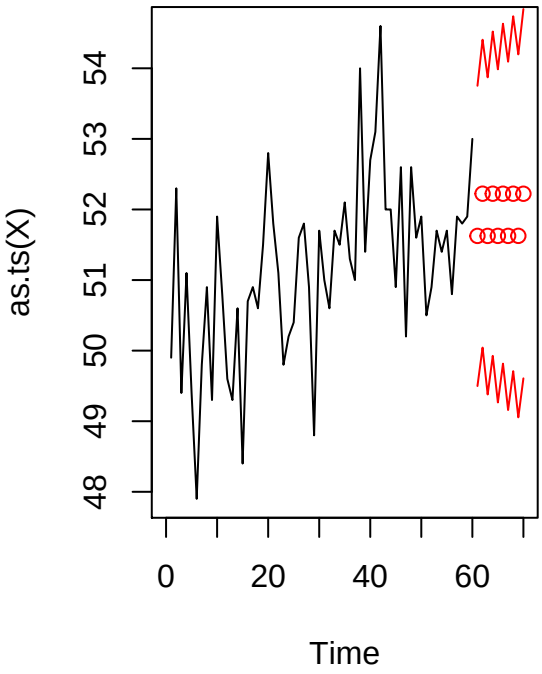



```
# test whether residuals are white noise by Ljung-Box
WNtest.LB(arima.out.AIC$resid,k = p.AIC + q.AIC + 1,nparms = p.AIC + q.AIC)
```

```
## [1] 0.09315422
```

```
# make predictions:
h <- 10
plot(as.ts(X),xlim=c(0,n+h))
predict.arima.out.AIC <- predict(arima.out.AIC,n.ahead=h)
X.hpred.arima.AIC <- predict.arima.out.AIC$pred
points(X.hpred.arima.AIC~c((n+1):(n+h)),col="red")

# put prediction intervals around predictions
```

```
X.hpred.se.arima.AIC <- predict.arima.out.AIC$se
lines(X.hpred.arima.AIC + 1.96 * X.hpred.se.arima.AIC ~ c((n+1):(n+h)),col="red")
lines(X.hpred.arima.AIC - 1.96 * X.hpred.se.arima.AIC ~ c((n+1):(n+h)),col="red")
```

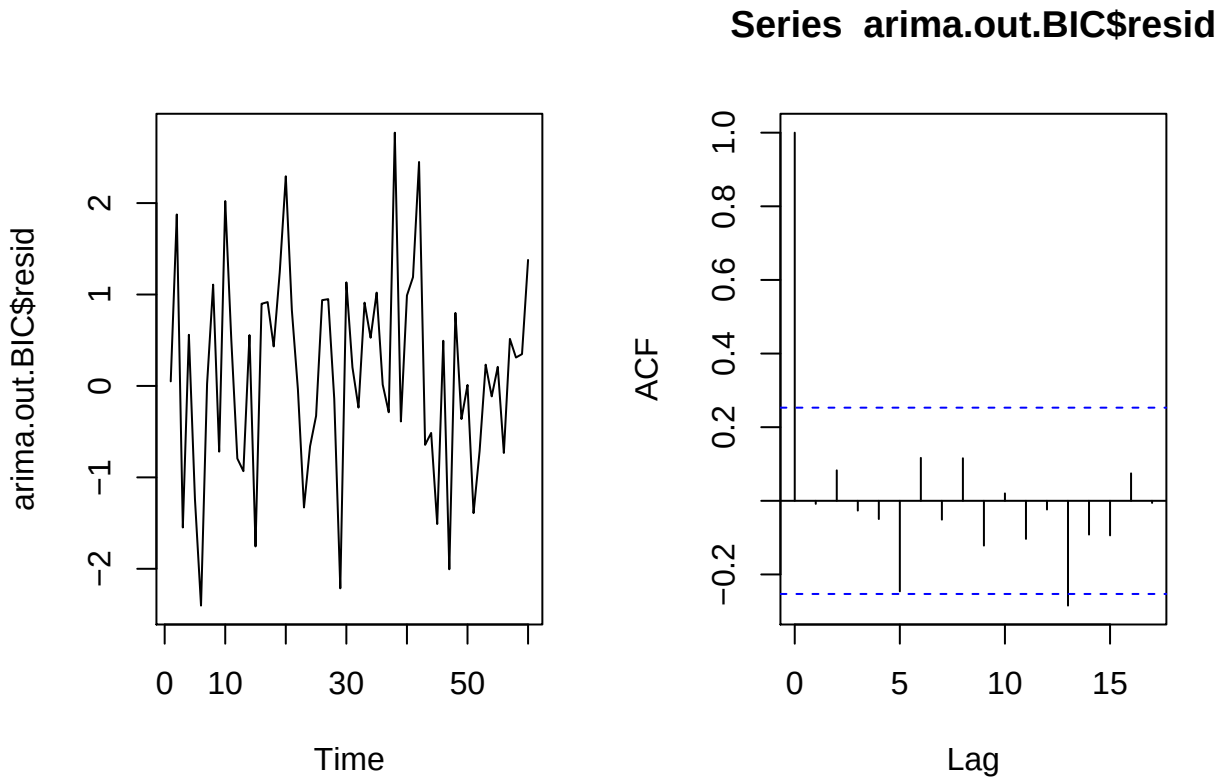


The predictions are very erratic because of the values of the fitted parameters. The next chunk of R code shows the model selected by the BIC criterion.

```
# identify model with lowest BIC
which.BIC <- which(BIC == min(BIC),arr.ind=TRUE)
p.BIC <- p.hat[which.BIC[1]]
q.BIC <- q.hat[which.BIC[2]]

# fit model with lowest BIC:
arima.out.BIC <- arima(X,order=c(p.BIC,1,q.BIC))
arima.out.BIC
```

```
##
## Call:
## arima(x = X, order = c(p.BIC, 1, q.BIC))
##
## Coefficients:
##           ma1
##       -0.7983
## s.e.   0.0956
##
## sigma^2 estimated as 1.291:  log likelihood = -91.76,  aic = 187.52
```

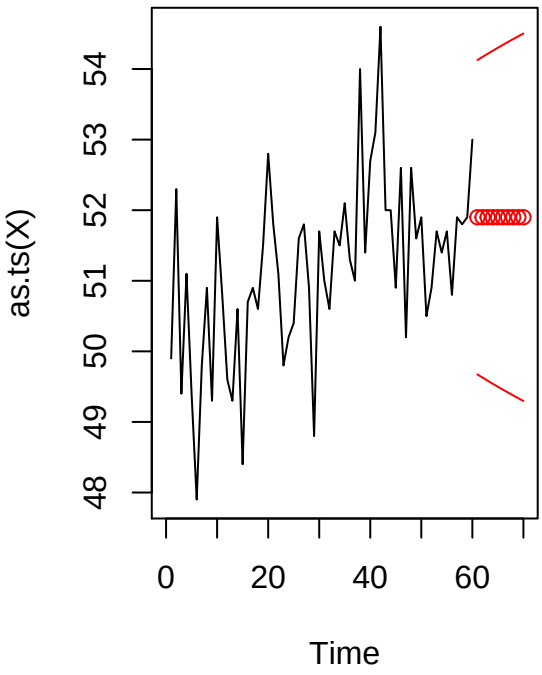
```
# look at residuals
par(mfrow=c(1,2))
plot(arima.out.BIC$resid)
acf(arima.out.BIC$resid)
```




```
# test whether residuals are white noise by Ljung-Box
WNtest.LB(arima.out.BIC$resid,k = p.BIC + q.BIC + 1,nparms = p.BIC + q.BIC)
```

```
## [1] 0.5050558
```

```
# make predictions:
h <- 10
plot(as.ts(X),xlim=c(0,n+h))
predict.arima.out.BIC <- predict(arima.out.BIC,n.ahead=h)
X.hpred.arima.BIC <- predict.arima.out.BIC$pred
points(X.hpred.arima.BIC~c((n+1):(n+h)),col="red")

# put prediction intervals around predictions
X.hpred.se.arima.BIC <- predict.arima.out.BIC$se
lines(X.hpred.arima.BIC + 1.96 * X.hpred.se.arima.BIC ~ c((n+1):(n+h)),col="red")
lines(X.hpred.arima.BIC - 1.96 * X.hpred.se.arima.BIC ~ c((n+1):(n+h)),col="red")
```



Dickey-Fuller unit root test

We may be interested in performing a statistical test to see whether a time series is non-stationary in the sense of having an ARMA representation such that the polynomial in the AR coefficients has a unit root. The Dickey-Fuller test tests this.

For AR(1) model

Consider for the moment an AR(1) model. Let $X_1, \ldots, X_n$ be a realization from the time series $\{X_t, t \in \mathbb{Z}\}$ satisfying

$$X_t - \mu = \phi(X_t - \mu) + Z_t,$$

where $\{Z_t, t \in \mathbb{Z}\}$ is WN$(0, \sigma^2)$ and $|\phi| < 1$.

We wish to test the hypotheses

$$H_0\colon\ \phi = 1 \text{ versus } H_1\colon\ \phi < 1.$$

So if we reject H_0, we can conclude that the time series is stationary.

If H_0 is true, it turns out that the maximum likelihood estimator $\hat{\phi}$ of $\phi = 1$ does not have a Normal limit distribution as $n \to \infty$; however, under H_1, under which $\phi \neq 1$, we have

$$\sqrt{n}(\hat{\phi} - \phi) \to \text{Normal}(0, 1 - \phi^2) \text{ in distribution}$$

as $n \to \infty$. But for hypothesis testing, it is the null distribution which is needed in order to calibrate the rejection region of the test according to the size of test desired. We therefore consider a different model.

Consider the time series given by $\nabla X_t = X_t - X_{t-1}$ for $t \in \mathbb{Z}$. We have

$$\begin{aligned}
\nabla X_t &= X_t - X_{t-1} \\
&= (X_t - \mu) - (X_{t-1} - \mu) \\
&= \phi(X_{t-1} - \mu) - (X_{t-1} - \mu) + Z_t \\
&= \underbrace{\mu(1-\phi)}_{\phi_0^*} + \underbrace{(\phi - 1)}_{\phi_1^*} X_{t-1} + Z_t \\
&= \phi_0^* + \phi_1^* X_{t-1} + Z_t,
\end{aligned}$$

where

$$\phi_0^* = \mu(1-\phi) \quad \text{and} \quad \phi_1^* = (\phi - 1).$$

Now, in order to test H_0: $\phi = 1$, we test H_0: $\phi_1^* = 0$. We can estimate ϕ_1^* in such a way that our estimator has a known distribution under the null hypothesis, which will allow us to calibrate a rejection region for the test according to any desired size.

We estimate ϕ_0^* and ϕ_1^* with the least squares estimators

$$\left(\hat{\phi}_0^*, \hat{\phi}_1^*\right) = \underset{\phi_0^*, \phi_1^* \in \mathbb{R}}{\operatorname{argmin}} \sum_{t=2}^{n} \left[\nabla X_t - (\phi_0^* + \phi_1^* X_{t-1})\right]^2.$$

We estimate the standard error of $\hat{\phi}_1^*$ with

$$\widehat{\mathrm{SE}}(\hat{\phi}_1^*) = \left(\frac{\frac{1}{n-1-2} \sum_{t=2}^{n} \left[\nabla X_t - (\hat{\phi}_0^* + \hat{\phi}_1^* X_{t-1})\right]^2}{\sum_{t=2}^{n} \left[X_{t-1} - \frac{1}{n-1} \sum_{j=1}^{n-1} X_j\right]^2} \right)^{1/2}.$$

Note that this is the same way in which we would estimate the standard error of the slope coefficient in simple linear regression, but we have $n-1$ instead of n data points because of the differencing.

The test statistic of the Dickey-Fuller test is defined as

$$\mathrm{DF} = \hat{\phi}_1^* / \widehat{\mathrm{SE}}(\hat{\phi}_1^*). \tag{3}$$

We will reject H_0: $\phi = 1$ in favor of H_1: $\phi < 1$ if the statistic is a large-enough negative value; if $\hat{\phi}_1^*$ is close to zero, than there is not strong evidence against H_0. At first glance, it seems that the null distribution of the Dickey-Fuller test statistic should be a t-distribution, but it is not exactly a t-distribution because of the relationship between the the response ∇X_t and the predictor X_{t-1} for $t = 2, \ldots, n$. The asymptotic (large n) critical values at the 0.01 and 0.05 levels are -3.43 and -2.86, respectively. See pg.~373 of Fuller (2009).

For AR(p) model

We can also test whether an AR(p) model has a unit root. Let $\{X_t, t \in \mathbb{Z}\}$ be the AR(p) process given by

$$(X_t - \mu) = \sum_{j=1}^{p} \phi_j (X_{t-j} - \mu) + Z_t, t \in \mathbb{Z},$$

where $\{Z_t, t \in \mathbb{Z}\}$ is WN$(0, \sigma^2)$. Consider checking whether the polynomial $\phi(u) = 1 - \phi_1 u - \cdots - \phi_p u^p$ has a unit root. If $\phi(u) = 0$ for $u = 1$, this means that $1 - \phi_1 - \cdots - \phi_p = 0$, or that the AR coefficients sum to one. This is what we will test:

$$H_0\text{: } \phi_1 + \cdots + \phi_p = 1 \text{ versus } H_1\text{: } \phi_1 + \cdots + \phi_p \neq 1$$

We again consider the time series $\nabla X_t = X_t - X_{t-1}$, $t \in \mathbb{Z}$. We can write

$$\begin{aligned}
\nabla X_t &= X_t - X_{t-1} \\
&= (X_t - \mu) - (X_{t-1} - \mu) \\
&= \phi_1(X_{t-1} - \mu) + \phi_2(X_{t-2} - \mu) + \cdots + \phi_p(X_{t-p} - \mu) - (X_{t-1} - \mu) + Z_t \\
&= \mu(1 - \phi_1 - \cdots - \phi_p) + (\phi_1 - 1)X_{t-1}\phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + Z_t \\
&= \mu(1 - \phi_1 - \cdots - \phi_p) + (\phi_1 + \cdots + \phi_p - 1)X_{t-1} + (-\phi_2 - \cdots - \phi_p)(X_{t-1} - X_{t-2}) + Z_t \\
&\quad \cdots + (-\phi_{p-1} - \phi_p)(X_{t-p+2} - X_{t-p+1}) + (-\phi_p)(X_{t-p+1} - X_{t-p}) + Z_t \\
&= \phi_0^* + \phi_1^* X_{t-1} + \phi_2^* \nabla X_{t-1} + \cdots + \phi_p^* \nabla X_{t-p+1} + Z_t,
\end{aligned}$$

where

$$\begin{aligned}
\phi_0^* &= \mu(1 - \phi_1 - \cdots - \phi_p) \\
\phi_1^* &= (\phi_1 + \cdots + \phi_p - 1) \\
\phi_2^* &= (-\phi_2 - \cdots - \phi_p) \\
&\vdots \\
\phi_{p-1}^* &= (-\phi_{p-1} - \phi_p) \\
\phi_p^* &= -\phi_p.
\end{aligned}$$

In order to test H_0: $\phi_1 + \cdots + \phi_p = 1$, we test H_0: $\phi_1^* = 0$. We estimate ϕ_1^* as before, but this time as a coefficient in a multiple linear regression model. That is we obtain

$$\left(\phi_0^*, \phi_1^* \ldots, \phi_p^*\right) = \underset{\phi_0, \phi_1, \ldots, \phi_p}{\operatorname{argmin}} \sum_{t=p+1}^{n} \left[\nabla X_t - (\phi_0^* + \phi_1^* X_{t-1} + \phi_2^* \nabla X_{t-1} + \ldots \phi_p^* \nabla X_{t-p+1})\right]^2.$$

We obtain an estimate $\widehat{\mathrm{SE}}(\hat{\phi}_1^*)$ of the standard error of $\hat{\phi}_1^*$ as we would in multiple linear regression. The Dickey-Fuller test statistic is given again by $\mathrm{DF} = \hat{\phi}_1^* / \widehat{\mathrm{SE}}(\hat{\phi}_1^*)$. We may compare this to the same critical values used for the AR(1) case.

The R code below defines a function for carrying out the Dickey-Fuller unit root test and runs the test on a simulated data set.

```
unitroottest.DF <- function(X,p=1)
{

    if(p==1)
    {

        Xdiff <- diff(X)
        Xlag <- X[1:(n-1)]
        lm.out <- lm(Xdiff ~ Xlag)
        DF <- summary(lm.out)$coef[2,3]

    } else if(p > 1){

    Xdiff <- diff(X)

    XX <- matrix(NA,n-p,p)
```

```
    XX[,1] <- X[p:(n-1)]

    for(j in 1:(p-1))
      {

        XX[,1+j] <- Xdiff[(p-j):(n-1-j)]

      }

      lm.out <- lm(Xdiff[p:(n-1)] ~ XX)
        DF <- summary(lm.out)$coef[2,3]

    }

    reject.01 <- FALSE
    reject.05 <- FALSE
    if(DF < -3.43)
    {
        reject.01 <- TRUE
    }
    if(DF < -2.86)
    {
        reject.05 <- TRUE

    }

    output <- list( reject.05 = reject.05,
                            reject.01 = reject.01,
                            DF = DF)

    return(output)

}

phi <- 0
theta <- NULL
sigma <- 2
n <- 200

X <- get.ARMA.data(phi,theta,sigma,n)
```

```
## Warning in min(Mod(polyroot(c(1, -phi)))): no non-missing arguments to min;
## returning Inf

## Warning in min(Mod(polyroot(c(1, -phi)))): no non-missing arguments to min;
## returning Inf
```

```
unitroottest.DF(X,p=5)
```

```
## $reject.05
## [1] TRUE
##
## $reject.01
## [1] TRUE
```

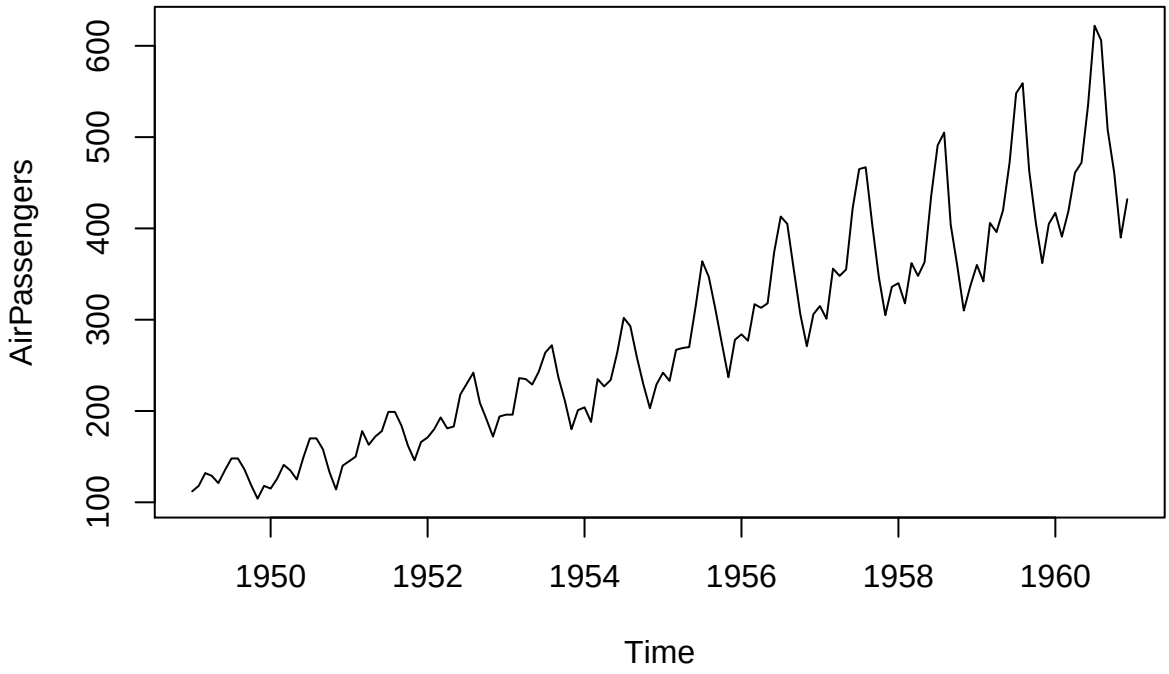
```
##
## $DF
## [1] -6.461597
```

SARIMA models

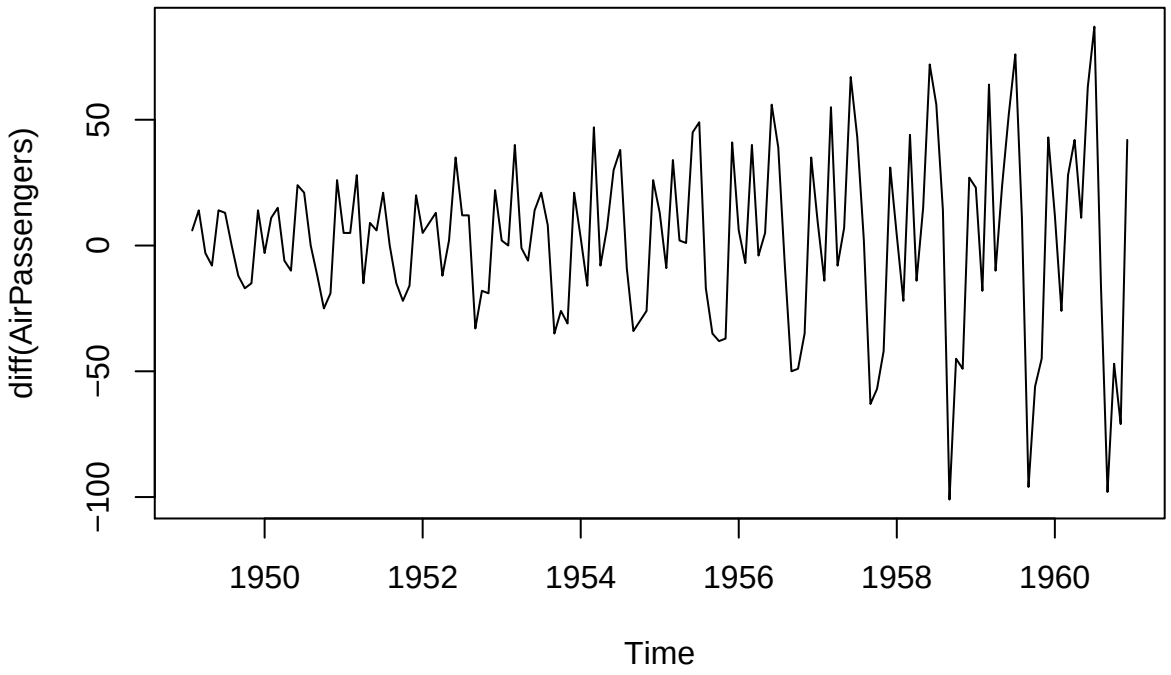
Seasonal ARIMA (SARIMA) models can accommodate unit-root-type non-stationarity as well as seasonality. Given a realization $X_1, \ldots, X_n$ from a time series $\{X_t, t \in \mathbb{Z}\}$, a SARIMA model is appropriate if for some $d \geq 1$, $D \geq 1$ and $s > 1$ the time series given by $Y_t = (1-B)^d(1-B^s)^D X_t$, $t \in \mathbb{Z}$, appears to be a stationary time series. The operation $(1-B)^d(1-B^s)^D$ takes differences at lag s a number of times given by D, which is intended to remove seasonal effects with period s, and then takes first-order differences a number of times equal to d, which is intended to remove a trend or unit-root-type non-stationarity.

The following R code applies to the `AirPassenger` data set from R the differencing $(1-B)(1-B^{12})$, corresponding to $d = 1$, $D = 1$, and $s = 12$, which results in a time series that appears to be stationary.

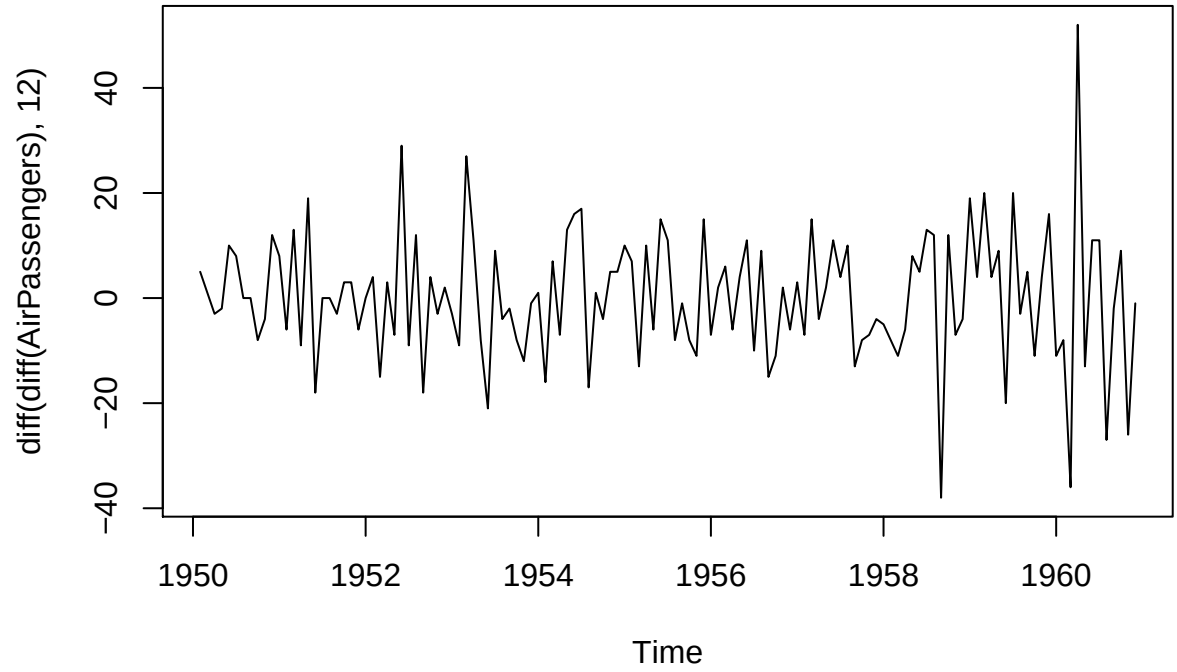
```
data(AirPassengers)
plot(AirPassengers)
```




```
plot(diff(AirPassengers))
```




```
plot(diff(diff(AirPassengers),12))
```




```
# Looks like d = 1 and D = 1.  Period of seasonality is s = 12.
```

Definition of a SARIMA process

A time series $\{X_t, t \in \mathbb{Z}\}$ is called an SARIMA$(p,d,q) \times (P,D,Q)_s$ process if the time series given by $Y_t = (1-B)^d(1-B^s)^D X_t$ for $t \in \mathbb{Z}$ is a causal ARMA process satisfying

$$\Phi(B^s)\phi(B)Y_t = \Theta(B^s)\theta(B)Z_t, \quad t \in \mathbb{Z},$$

where $\{Z_t, t \in \mathbb{Z}\}$ is WN$(0, \sigma^2)$ and where

$$\Phi(u) = 1 - \Phi_1 u - \cdots - \Phi_P u^P \quad \text{and} \quad \phi(u) = 1 - \phi_1 u - \cdots - \phi_p u^p$$
$$\Theta(u) = 1 - \Theta_1 u - \cdots - \Theta_Q u^Q \quad \text{and} \quad \theta(u) = 1 - \theta_q u - \cdots - \theta_q u^q.$$

Note that for $\{Y_t, t \in \mathbb{Z}\}$ to be causal, we require $\Phi(u) \neq 0$ and $\phi(u) \neq 0$ for all $|u| \leq 1$.

To understand the definition of the SARIMA model, consider arranging a realization $Y_1, \ldots, Y_n$ of the time series $Y_t = (1-B)^d(1-B^s)^D X_t$ in a table as follows:

	1	2	...	s
1	Y_1	Y_2	...	Y_s
2	Y_{s+1}	Y_{s+2}	...	Y_{2s}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
k	$Y_{(k-1)s+1}$	Y_{ks+2}	...	Y_{ks}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
N	$Y_{(N-1)s+1}$	$Y_{(N-1)s+1}$	...	Y_{Ns}

Now we consider the dependence within the columns of the above table to be that of an ARMA(P, Q) process; that is, we assume

$$Y_t = \Phi_1 Y_{t-s} + \ldots \Phi_P Y_{t-sP} + \Theta_1 U_{t-s} + \cdots + \Theta_Q U_{t-sQ} + U_t, \quad t \in \mathbb{Z}$$

for some time series $\{U_t, t \in \mathbb{Z}\}$. We may write the above as

$$\Phi(B^s) Y_t = \Theta(B^s) U_t, \quad t \in \mathbb{Z}.$$

If we in turn assume that the time series $\{U_t, t \in \mathbb{Z}\}$ is an ARMA(p, q) process satisfying

$$\phi(B) U_t = \theta(B) Z_t, \quad t \in \mathbb{Z},$$

where $\{Z_t, t \in \mathbb{Z}\}$ is WN$(0, \sigma^2)$, we have

$$\Phi(B^s) Y_t = \Theta(B^s) U_t \iff \Phi(B^s)\phi(B) Y_t = \Theta(B^s)\phi(B) U_t.$$

Lastly, since $\phi(B) U_t = \theta(B) Z_t$, we have

$$\Phi(B^s)\phi(B) Y_t = \Theta(B^s)\theta(B) Z_t, \quad t \in \mathbb{Z}.$$

Note that the polynomials given by $\phi^*(u) = \Phi(u^s)\phi(u)$ and $\theta^*(u) = \Theta(B^s)\theta(B)$ are of order $p + sP$ and $q + sQ$, respectively.

Estimation and forecasting with SARIMA models

We can estimate the parameters of the SARIMA$(p, d, q) \times (P, D, Q)_s$ model from some data $X_1, \ldots, X_n$ by obtaining the values $Y_t = (1-B)^d(1-B^s)^D X_t$ and then computing the maximum likelihood estimators of the ARMA model

$$\phi^*(B) Y_t = \theta^*(B) Z_t, \quad t \in \mathbb{Z},$$

where $\{Z_t, t \in \mathbb{Z}\}$ is WN$(0, \sigma^2)$ and $\phi^*(\cdot)$ and $\theta^*(\cdot)$ are the polynomials given by $\phi^*(u) = \Phi(u^s)\phi(u)$ and $\theta^*(u) = \Theta(B^s)\theta(B)$. These can be computed easily using the`arima()` function in R.

The following shows how to fit a SARIMA$(p, d, q) \times (P, D, Q)_s$ on the `AirPassenger` data.

```r
# Example of how to fit a SARIMA model:
s <- 12

p <- 1
d <- 1
q <- 1

P <- 1
D <- 1
Q <- 1

arima.out <- arima(AirPassengers,
                   order=c(p,d,q),
                   seasonal=list(order=c(P,D,Q),period=s))
arima.out
```

```
##
## Call:
## arima(x = AirPassengers, order = c(p, d, q), seasonal = list(order = c(P, D,
##     Q), period = s))
##
## Coefficients:
##           ar1      ma1     sar1    sma1
##       -0.1386  -0.2028  -0.9228  0.8329
## s.e.   0.5865   0.6128   0.2387  0.3519
##
## sigma^2 estimated as 130.8:  log likelihood = -506.15,  aic = 1022.3
```

Just as we chose the orders p and q for ARIMA(p, d, q) model using the AIC and BIC criteria, we can also choose the orders p, q, P, and Q of the SARIMA$(p, d, q) \times (P, D, Q)_s$ model using the AIC and BIC. We will suppose that d, D, and s can be found by taking differences as looking at plots of the differenced series. The period s will in many cases be evident from context of the data.

The R code below illustrates using the AIC and BIC to choose p, q, P, and Q of the SARIMA$(p, d, q) \times (P, D, Q)_s$ model on the `AirPassengers` data set from R. In addition, it shows how to use the `predict.Arima()` function to get h-step-ahead predictions from the fitted SARIMA model for $h \geq 1$.

```r
# Now choose orders p, q, P, and Q by AIC or BIC. The number of times to difference,
# d and D, can be ascertained by taking differences and looking at plots of the
# differenced series, as above.

p.hat <- 0:2
q.hat <- 0:2
P.hat <- 0:2
Q.hat <- 0:2

AIC <- BIC <- array(NA,dim=c(length(p.hat),length(q.hat),length(P.hat),length(Q.hat)))
for( j in 1:length(p.hat))
  for( k in 1:length(q.hat))
    for( l  in 1:length(P.hat))
      for( m in 1:length(Q.hat))
      {
        # use the try() function because there might be convergence issues for some
        # models with a larger number of parameters
        try({
```

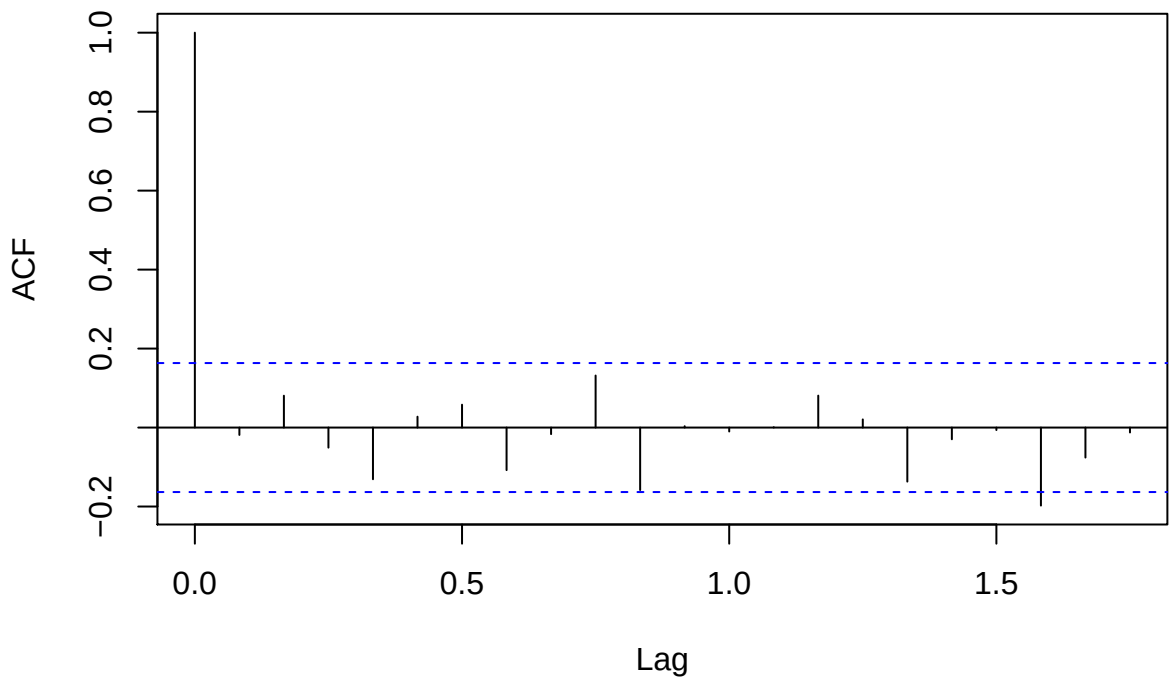
```
              arima.out.jklm <- arima(AirPassengers,
                                      order=c(p.hat[j],d,q.hat[k]),
                                      seasonal=list(order=c(P.hat[l],D,Q.hat[m]),period=s))
              AIC[j,k,l,m] <- AIC(arima.out.jklm)
              BIC[j,k,l,m] <- BIC(arima.out.jklm)
            })
      }
```

```
## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in arima(AirPassengers, order = c(p.hat[j], d, q.hat[k]), seasonal
## = list(order = c(P.hat[l], : possible convergence problem: optim gave code
## = 1

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in arima(AirPassengers, order = c(p.hat[j], d, q.hat[k]), seasonal
## = list(order = c(P.hat[l], : possible convergence problem: optim gave code
## = 1

## Warning in log(s2): NaNs produced
```

```
# look at AIC-selected model
ind.AIC <- which(AIC == min(AIC,na.rm=TRUE), arr.ind=TRUE)
p.AIC <- p.hat[ind.AIC[1]]
q.AIC <- q.hat[ind.AIC[2]]
P.AIC <- P.hat[ind.AIC[3]]
Q.AIC <- Q.hat[ind.AIC[4]]

arima.AIC <- arima(AirPassengers,
                   order=c(p.AIC,d,q.AIC),
                   seasonal=list(order=c(P.AIC,D,Q.AIC),period=s))
arima.AIC
```

```
##
## Call:
## arima(x = AirPassengers, order = c(p.AIC, d, q.AIC), seasonal = list(order = c(P.AIC,
##     D, Q.AIC), period = s))
##
## Coefficients:
##           ar1     ma1      ma2    sar1     sma1    sma2
##       -0.9187  0.5407  -0.4587  0.9824  -1.3112  0.4014
## s.e.   0.0459  0.0886   0.0876  0.0594   0.1950  0.1192
##
## sigma^2 estimated as 107:  log likelihood = -499.16,  aic = 1012.32
```

```
# look at residuals of fitted model; check to see if they are white noise
acf(arima.AIC$resid)
```




```
nparms.AIC <- sum(p.AIC,q.AIC,P.AIC,Q.AIC,1)
# use a built-in function for the Ljung-Box test
Box.test(arima.AIC$resid,type="Ljung-Box", fitdf = nparms.AIC, lag = nparms.AIC + 1)
```

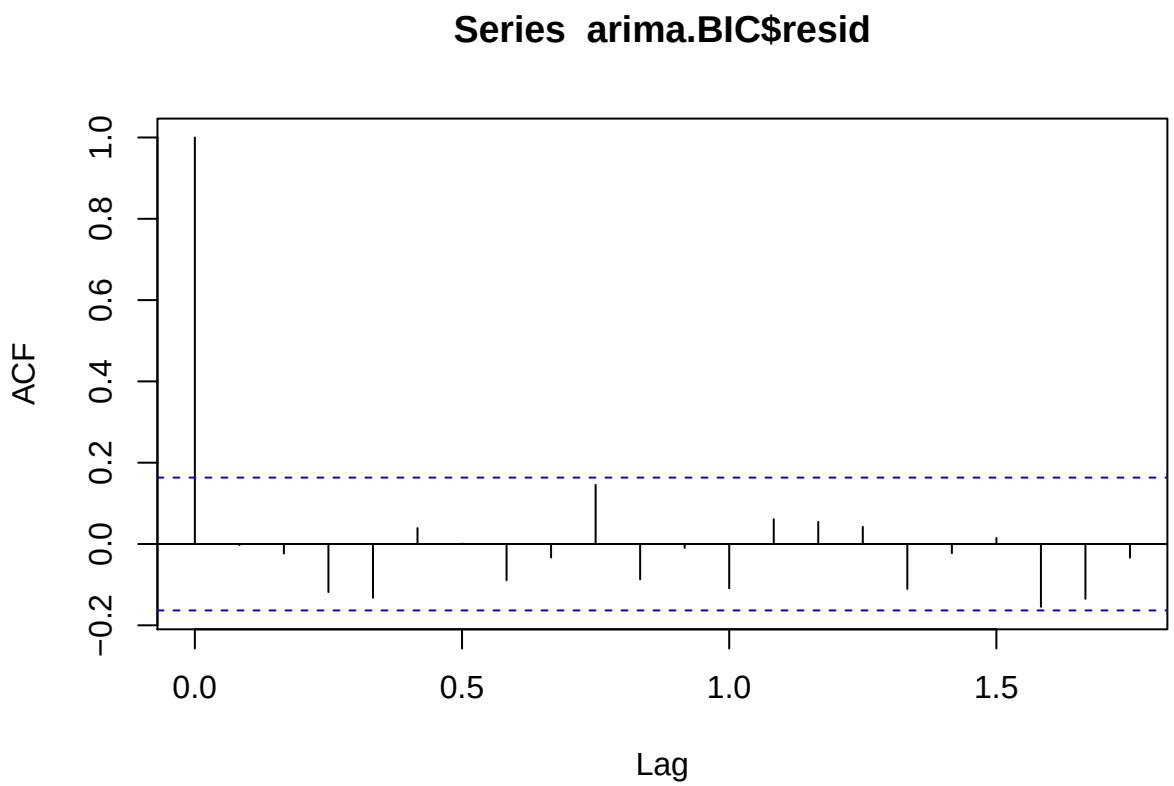
```
##
##  Box-Ljung test
##
## data:  arima.AIC$resid
## X-squared = 6.4264, df = 1, p-value = 0.01124
```

```
# look at BIC-selected model
ind.BIC <- which(BIC == min(BIC,na.rm=TRUE), arr.ind=TRUE)
p.BIC <- p.hat[ind.BIC[1]]
q.BIC <- q.hat[ind.BIC[2]]
P.BIC <- P.hat[ind.BIC[3]]
Q.BIC <- Q.hat[ind.BIC[4]]

arima.BIC <- arima(AirPassengers,
                   order=c(p.BIC,d,q.BIC),
                   seasonal=list(order=c(P.BIC,D,Q.BIC),period=s))
arima.BIC
```

```
##
## Call:
## arima(x = AirPassengers, order = c(p.BIC, d, q.BIC), seasonal = list(order = c(P.BIC,
##     D, Q.BIC), period = s))
##
## Coefficients:
##           ar1
##       -0.3076
```

```
## s.e.   0.0828
##
## sigma^2 estimated as 137:  log likelihood = -508.2,  aic = 1020.39
```

```
# look at residuals of fitted model; check to see if they are white noise
acf(arima.BIC$resid)
```

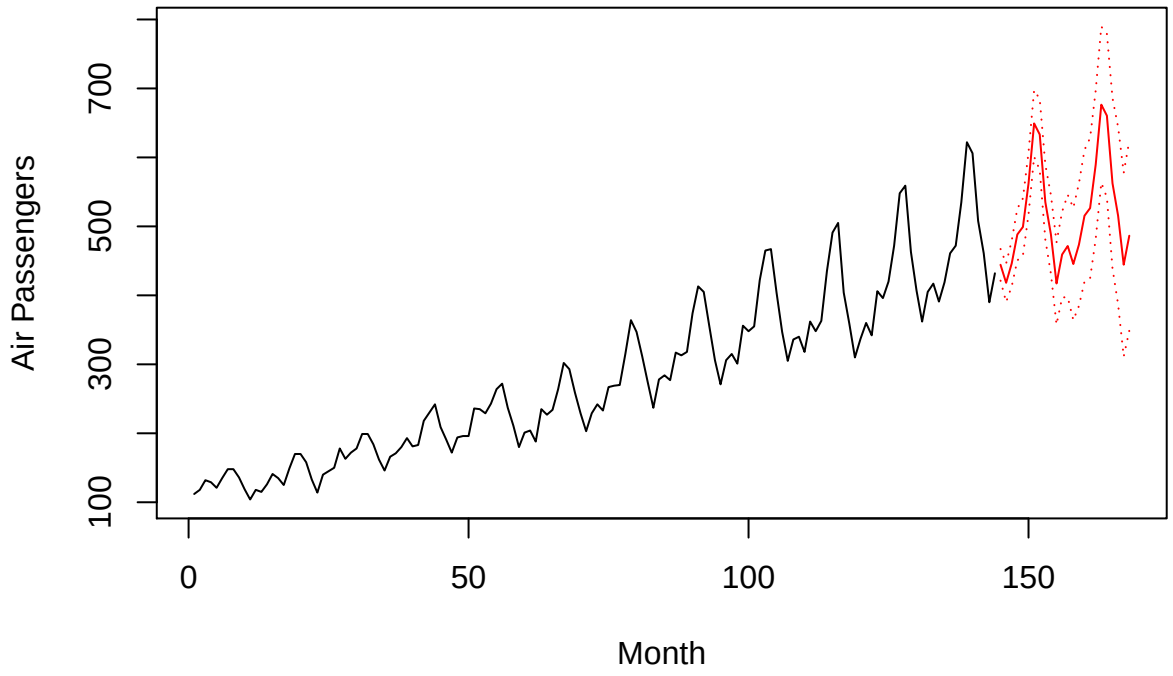



```
nparms.BIC <- sum(p.BIC,q.BIC,P.BIC,Q.BIC,1)
Box.test(arima.BIC$resid,type="Ljung-Box", fitdf = nparms.BIC, lag = nparms.BIC + 1)
```

```
##
##  Box-Ljung test
##
## data:  arima.BIC$resid
## X-squared = 2.1689, df = 1, p-value = 0.1408
```

```
# fun part: get predictions from BIC-selected SARIMA model
n <- length(AirPassengers)
h <- 24
predict.arima.BIC <- predict(arima.BIC,n.ahead=h)
pred <- predict.arima.BIC$pred
upper <- predict.arima.BIC$pred + 1.96 * predict.arima.BIC$se
lower <- predict.arima.BIC$pred - 1.96 * predict.arima.BIC$se

plot(as.numeric(AirPassengers),
     xlim=c(1,(n+h)),
     type="l",
     ylim=range(AirPassengers,upper,lower),
     ylab="Air Passengers",
     xlab="Month")
```

```r
lines(predict.arima.BIC$pred~c((n+1):(n+h)),col="red")
upper <- predict.arima.BIC$pred + 1.96 * predict.arima.BIC$se
lower <- predict.arima.BIC$pred - 1.96 * predict.arima.BIC$se

lines(lower~c((n+1):(n+h)),col="red",lty=3)
lines(upper~c((n+1):(n+h)),col="red",lty=3)
```



References

Fuller, Wayne A. 2009. *Introduction to Statistical Time Series*. Vol. 428. John Wiley & Sons.