COSC/DATA 405/505

Modelling and Simulation

Outline

Autoregressive Processes of Order 1 (e.g. Flood Risk)

The Markov Property

Autocovariance and Autocorrelation

Higher Order Autoregressive Processes, (e.g. Population Ecology)

MA, ARMA and ARIMA Processes (e.g. Climate Change)

ARCH Processes (e.g. FTSE Stock Data)

An autoregressive time series model

Suppose ε_1 and Z_0 are independent normal random variables, and suppose ϕ_1 is a constant. Suppose the expected value of ε_1 is 0, and the variance is $\sigma_\varepsilon^2 > 0$. $\mu_{Z_0} = E[Z_0]$ and $\sigma_{Z_0}^2 = \mathsf{Var}(Z_0)$. Let

$$Z_1 = \phi_1 Z_0 + \varepsilon_1.$$

It is possible to show that Z_1 is normally distributed, and using the earlier results on expected value and variance, Z_1 has mean $\phi_1 \mu_{Z_0}$ and variance $\sigma_{Z_0}^2 \phi_1^2 + \sigma_\varepsilon^2$.

An autoregressive time series model - stationarity

We now want to find conditions on ϕ_1 and μ_{Z_0} so that the distribution of Z_1 will be exactly the same as the distribution of Z_0. This kind of stationarity condition is often useful in modelling of processes that occur in time (or in space, for that matter).

$$E[Z_0] = \mu_{Z_0} = E[Z_1] = \phi_1 \mu_{Z_0}$$

implies that either $\phi_1 = 1$ or $\mu_{Z_0} = 0$.

$$V(Z_0) = \sigma^2_{Z_0} = V(Z_1) = \sigma^2_{\varepsilon} + \phi_1^2 \sigma^2_{Z_0}.$$

If $\phi_1 = 1$, then $\sigma_{\varepsilon} = 0$, which is not possible. Therefore, $\phi_1 \neq 1$. This means $\mu_{Z_0} = 0$.

An autoregressive time series model - stationarity

But we also have

$$\sigma^2_{Z_0} = \sigma^2_\varepsilon + \sigma^2_{Z_0}\phi_1^2$$

so that

$$\sigma^2_{Z_0}(1 - \phi_1^2) = \sigma^2_\varepsilon$$

which implies that $\phi_1^2 < 1$, and

$$\sigma^2_{Z_0} = \frac{\sigma^2_\varepsilon}{1 - \phi_1^2}.$$

An autoregressive time series model

Summarizing the results of the example, we observe that if Z_0 has a normal distribution with mean 0 and variance $\frac{\sigma_\varepsilon^2}{1-\phi_1^2}$, independent of ε_1 which also has a normal distribution with mean

0 and variance σ_ε^2, then

$$Z_1 = \phi_1 Z_0 + \varepsilon_1$$

has the same distribution as Z_0.

An autoregressive time series model

Now, let ε_2 be another normal random variable, independent of ε_1 but with the same mean and variance. Then

$$Z_2 = \phi_1 Z_1 + \varepsilon_2$$

must have the same distribution as Z_1.

In fact, for $n = 2, 3, \ldots$,

$$Z_n = \phi_1 Z_{n-1} + \varepsilon_n$$

defines a sequence of normal random variables all having mean 0 and variance $\frac{\sigma_\varepsilon^2}{1-\phi_1^2}$, when $\phi_1^2 < 1$ and the ε's are independent of each other.

The Z's have the same distribution but are dependent. This is a very important form of dependence: the Z's form a stationary Markov process.

The Markov Property

Recall that if a sequence of random variables $Z_1, Z_2, \ldots, Z_n$ is independent, then their joint distribution can be factored:

$$f(z_1, z_2, \ldots, z_n) = f(z_1)f(z_2)\cdots f(z_n).$$

If the sequence is completely dependent, the factorization involves complicated conditional distributions along the lines of:

$$f(z_1, z_2, \ldots, z_n) = f(z_n|z_1, z_2, \ldots, z_{n-1})f(z_{n-1}|z_1, \ldots, z_{n-2})\cdots$$

$$\cdots f(z_4|z_1, z_2, z_3)f(z_3|z_1, z_2)f(z_2|z_1)f(z_1).$$

Back in the 19th century, A. Markov realized that there might be a useful class of models in between these two extremes.

The Markov Property

$f(z_3|z_1, z_2)$ can be approximated by $f(z_3|z_2)$. This will usually be a better approximation than $f(z_3)$ only - which is what independence implies. This approximation says that z_3 is dependent on z_1 only through z_2 explicitly. In other words, given z_2, no additional information in the sample will provide information about z_3.

Similarly, approximate $f(z_4|z_1, z_2, z_3)$ by the simpler $f(z_4|z_3)$, and so on, finally approximating $f(z_n|z_1, \ldots, z_{n-1})$ by $f(z_n|z_{n-1})$.

That is, assume

$$f(z_1, z_2, \ldots, z_n) = f(z_n|z_{n-1})f(z_{n-1}|z_{n-2})\cdots$$

$$\cdots f(z_4|z_3)f(z_3|z_2)f(z_2|z_1)f(z_1).$$

... This is the Markov property.

The Lag 1 Autocovariance for AR(1) Data

We saw the use of autocorrelations and autocovariances in detecting (linear) dependence problems in random number generation. These tools are helpful in the study of linear time series models.

Recall that the covariance of X and Y is

$$E[XY] - E[X]E[Y].$$

Since

$$E[Z_n] = 0$$

the covariance of Z_n and Z_{n-1}, also called the lag 1 autocovariance is

$$\gamma_1 = E[Z_n Z_{n-1}]$$

$$= E[(\phi_1 Z_{n-1} + \varepsilon_n) Z_{n-1}] =$$

$$= \phi_1 E[Z_{n-1}^2] + 0$$

$$= \phi_1 \sigma_Z^2.$$

The Lag 1 Autocorrelation for AR(1) Data

Recall: the correlation is defined as the covariance of X and Y divided by the product of the standard deviations of X and Y.

For a stationary AR(1) process, the standard deviations of Z_{n-1} and Z_n are the same, so we divide by the variance of Z_n to get the lag 1 autocorrelation:

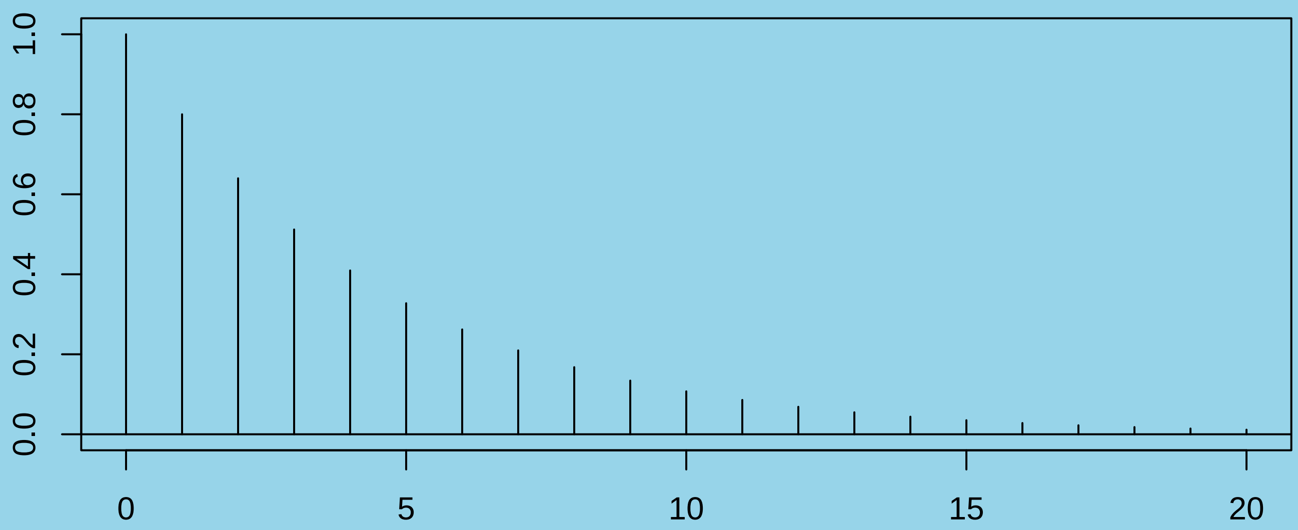
$$\rho_1 = \frac{\gamma_1}{\mathsf{Var}(Z_n)} = \phi_1.$$

Similar reasoning $\rightsquigarrow$

$$\rho_k = \phi_1^k$$

The Theoretical Autocorrelation Function for the AR(1) Process

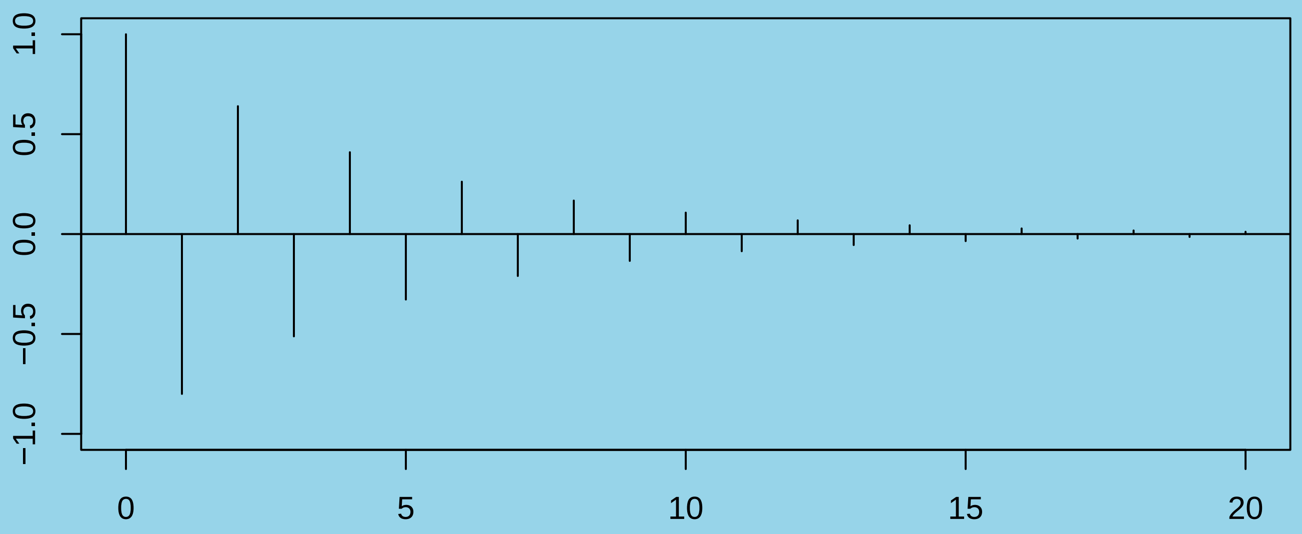
Here is a plot of the autocorrelation function for an AR(1) process with $\phi_1 = .8$:



All data points are correlated with each other, but dependence decays exponentially with increasing time between the points.

The Theoretical Autocorrelation Function for the AR(1) Process

Here is a plot of the autocorrelation function for an AR(1) process with $\phi_1 = -.8$:



Again, the correlations are all nonzero, but this time each observation is negatively correlated with the previous observation.

An autoregressive time series model - with nonzero mean

Usually, the expected value or mean level of the process is some nonzero value μ, so this value is usually subtracted from the time series.

That is, $Z_n = Y_n - \mu$, where Y_n is the original time

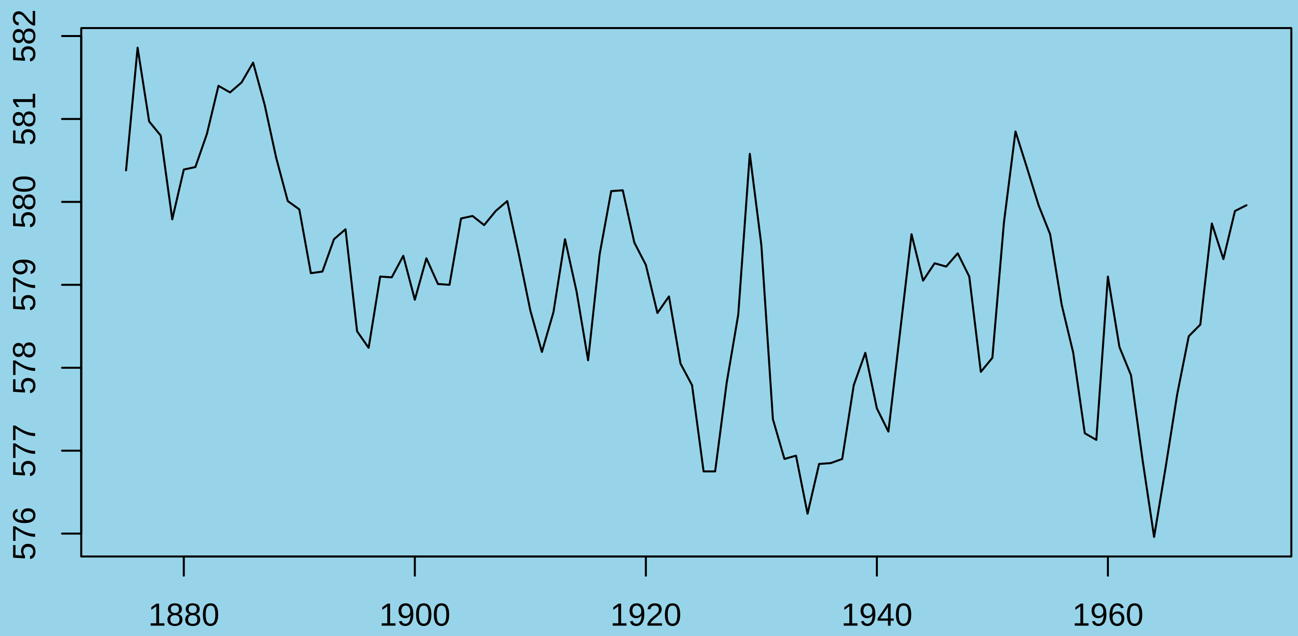
series, having expected value $E[Y_n] = \mu$.

Writing the autoregressive model in terms of Y's, we have

$$Y_n = \mu + \phi_1(Y_{n-1} - \mu) + \varepsilon_n.$$

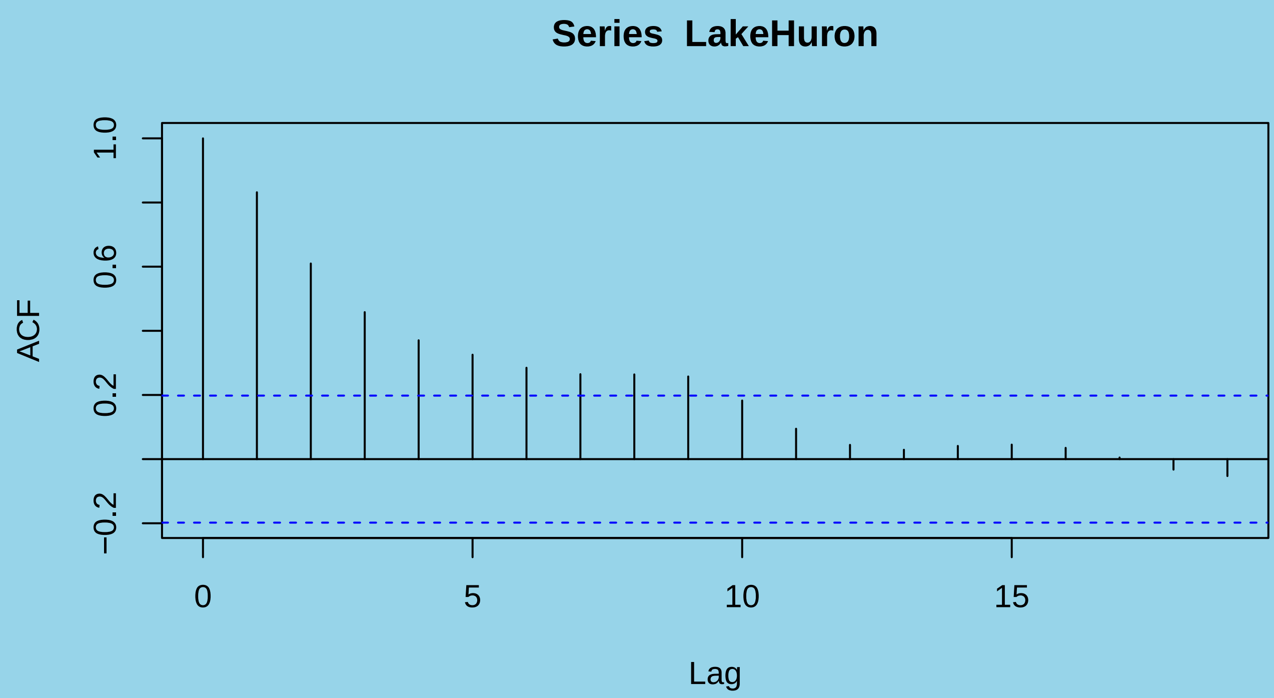
Example

We consider the annual levels of Lake Huron in the `LakeHuron` data object.



Example - Sample ACF

```r
acf(LakeHuron)
```



The sample ACF looks very similar to the theoretical ACF for an AR(1) process with positive ϕ_1.

Example - Fitting an AR(1) Model

The parameters of the AR(1) model can be estimated by maximum likelihood estimation with the arima() function:

```
lh <- arima(LakeHuron, order=c(1,0,0))
lh

##
## Call:
## arima(x = LakeHuron, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.8375   579.1153
## s.e.  0.0538     0.4240
##
## sigma^2 estimated as 0.5093:  log likelihood = -106.6,  aic = 219.2
```

The first component of the order parameter specifies the order of the autoregressive model: 1.

The other two components are 0, in this case. We will see their use later.

Example

The output from the function tells us that the ϕ_1 parameter is estimated as 0.8375, and μ is estimated at 579.1153.

Standard error estimates are provided for these estimates as well.

Both are small relative to the parameter estimates indicating that we have a fair bit of precision.

The variance σ^2 is also estimated as 0.5093.

Predicting with the Fitted Model

The fitted autoregressive model is

$$\widehat{Y}_n = 579.1 + .8375(Y_{n-1} - 579.1)$$

By plugging in a value for Y_{n-1}, we can use the fitted value to predict Y_n. The variance of the prediction can be calculated easily as well which gives us a standard error for the prediction (when we take a square root).

This is all coded in the `predict.Arima()` function.

Predicting with the Fitted Model

For example, to predict the next 5 years of Lake Huron levels, use

```
predict(lh, n.ahead=5)

## $pred
## Time Series:
## Start = 1973
## End = 1977
## Frequency = 1
## [1] 579.8228 579.7078 579.6116 579.5310 579.4634
##
## $se
## Time Series:
## Start = 1973
## End = 1977
## Frequency = 1
## [1] 0.7136434 0.9308795 1.0569470 1.1370689 1.1900573
```

Simulating autoregressive time series of order 1

Once we have the estimated values of all parameters in an autoregressive time series model, simulation is straightforward using a `for()` loop.

A starting value Z_0 is needed.

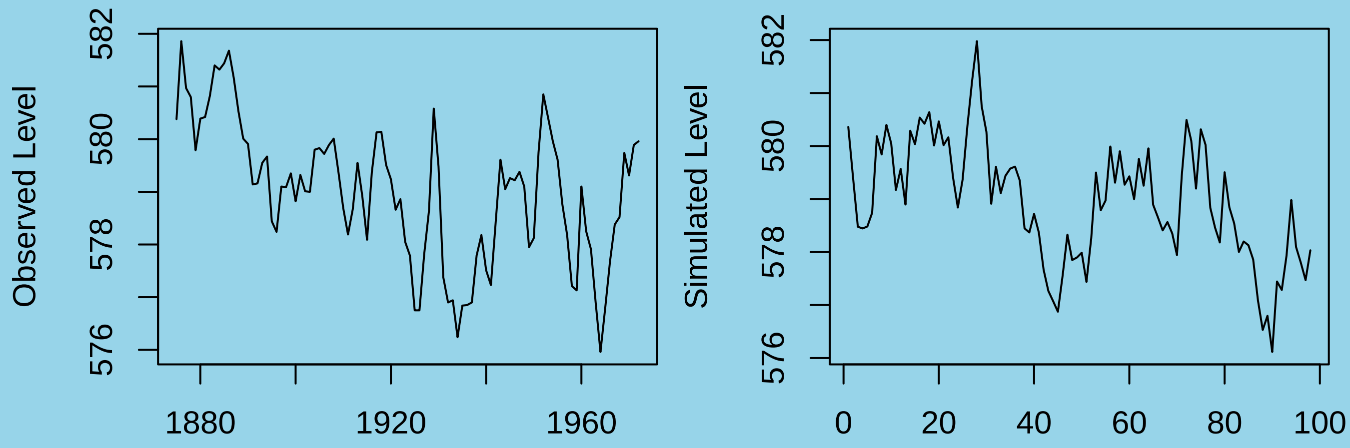
The first observed data point could serve this purpose, or the value could be simulated from a normal distribution with mean 0 and variance $\sigma^2/(1-\phi_1^2)$.

Example

Let's simulate from the fitted model for the Lake Huron data.

```
n <- length(LakeHuron); phi1 <- .8375; sigma2 <-.5093
Z0 <- rnorm(1, mean = 0, sd = sqrt(sigma2/(1 - phi1^2)))
epsilon <- rnorm(n, sd = sqrt(sigma2))
Z <- as.numeric(n)
Z[1] <- phi1*Z0 + epsilon[1]
for (i in 2:n) {
    Z[i] <- phi1*Z[i-1] + epsilon[i]
}
mu <- 579.1153
SimLake <- Z + mu  # add back the mean level
```

Graphing the Simulated Data



Left panel: Lake Huron levels for the years 1875 through 1972. Right panel: 98 years of data simulated from an autoregressive process designed to mimic the behaviour of the Lake Huron levels.

Simulating with the Built-In Function

The arima.sim() function can also be used to simulate autoregressive time series data.

This function takes several arguments, including `n` to specify the number of elements of the series, and `model` to specify the parameters of the model, using a `list`.

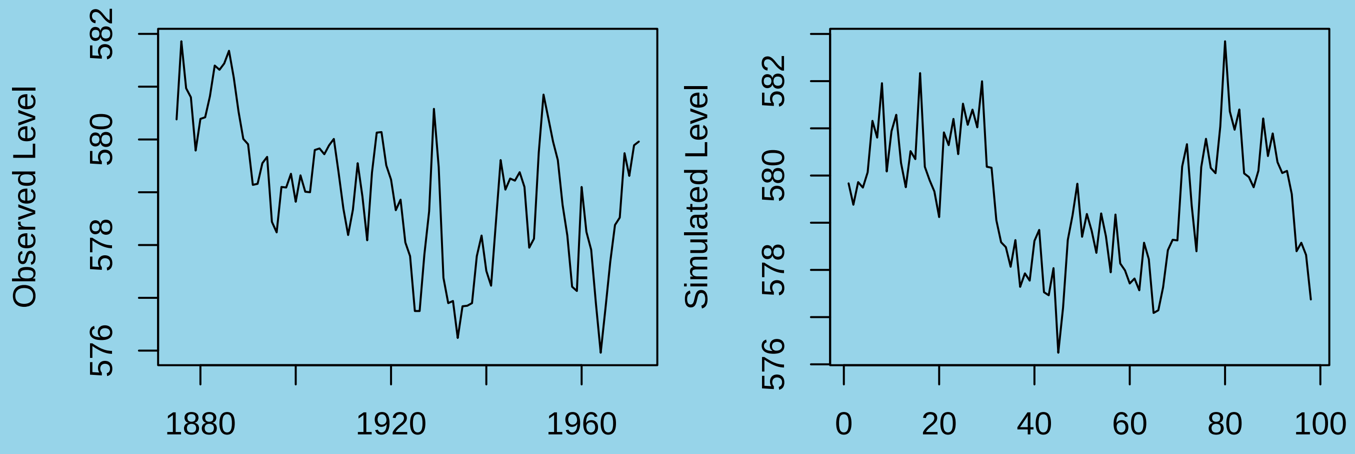
This list can include an element called `ar` which contains the autoregressive parameters and `sd` which contains the standard deviation of the noise.

Example

Simulating the AR(1) process mimicking the Lake Huron levels runs as follows:

```r
Z <- arima.sim(98, model=list(ar=phi1, sd=sqrt(sigma2)))
SimLake2 <- Z + mu  # add back the mean level
```

Graphing the Simulated Data



Left panel: Lake Huron levels for the years 1875 through 1972.

Right panel: 98 years of data simulated from an autoregressive process designed to mimic the behaviour of the Lake Huron levels, arising from the arima.sim function.

Higher Order Autoregressive Processes

Another example of a stationary time series model is the autoregressive order 2 (AR(2)) process:

$$Z_n = \phi_1 Z_{n-1} + \phi_2 Z_{n-2} + \varepsilon_n$$

as long as $|\phi_2| < 1$, $\phi_1 + \phi_2 < 1$ and $\phi_2 - \phi_1 < 1$.

Including the mean level, the model becomes

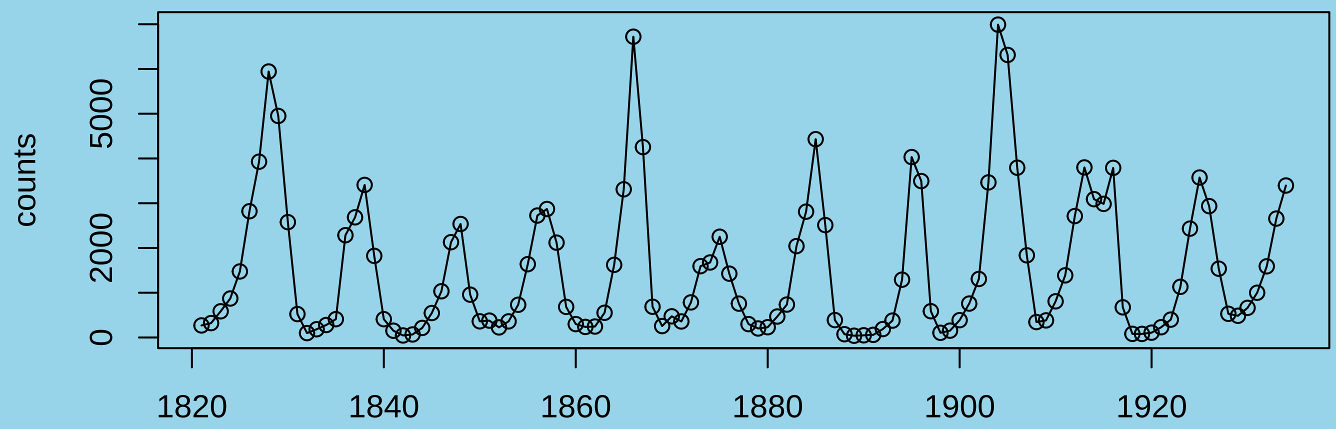
$$Y_n = \mu + \phi_1(Y_{n-1} - \mu) + \phi_2(Y_{n-2} - \mu) + \varepsilon_n.$$

Higher order AR processes can be considered as well, where Y_n depends on terms involving Y_{n-3} and so on.

Example: Annual Canadian Lynx

The data in the `lynx` object concern the annual numbers of lynx trapped in northern Canada for the years 1821 through 1934. The next figure contains a trace plot of these counts, produced from the following code:

```
ts.plot(lynx) # draw a broken line curve through the data points
points(1821:1934, (lynx)) # include the data points on the plot
```



Example: Annual Canadian Lynx

What is obvious on the plot is the periodic behaviour.

Every few years the counts dramatically increase before just as dramatically collapsing, almost to 0, remaining at that level for awhile before repeating the cycle.

What is also evident on the plot, though not quite as obviously, is that the variability of the counts changes, depending upon the stage of the cycle.

Note, in particular, that when the troughs of the curve are at very similar levels, but the peaks of the curve are highly variable. This is an indicator that the variability is not constant.

Use a Square Root Transformation when Analyzing Counts

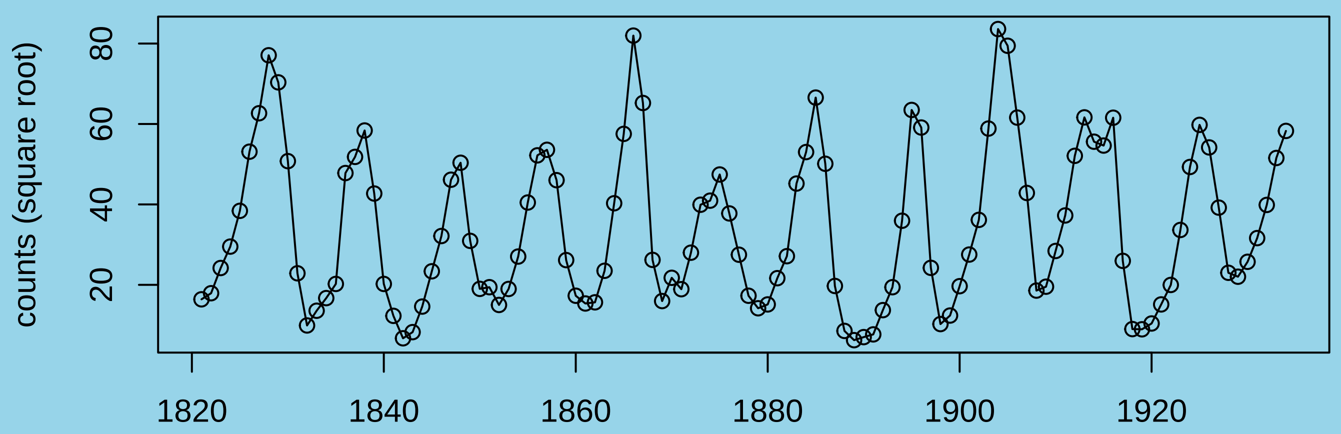
With count data, it is often a good idea to work with square roots of the counts.

This kind of transformation has the effect of substantially reducing very large values while having less effect on small and moderate values: this is a form of {variance-stabilizing transformation.

Use of the Square Root Transformation

The next figure shows the effect of taking the square root on each of the counts.

```
ts.plot(sqrt(lynx))
points(1821:1934, sqrt(lynx))
```



Use of the Square Root Transformation

Now the variability of the troughs is larger while the variability of the peaks is slightly reduced.

Overall the variation is about the same, no matter what stage of the cycle one is looking at.

Exploring the Sample ACF

Autocorrelations at lags 0, 1 and 2

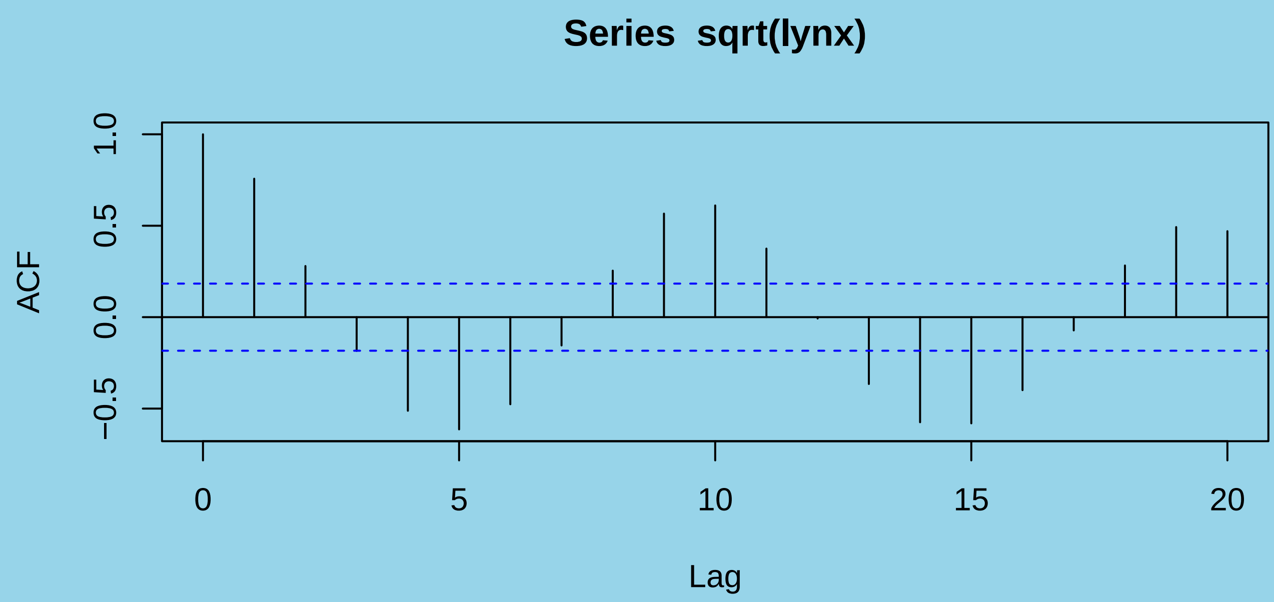
```
acf(sqrt(lynx), plot=FALSE)$acf[1:3]

## [1] 1.0000000 0.7571939 0.2796137
```

and at lags 3, 4 and 5

```
acf(sqrt(lynx), plot=FALSE)$acf[4:6]

## [1] -0.1843740 -0.5118717 -0.6137455
```

Neighbouring values are highly correlated. Values separated by 5 or 6 time units are negatively correlated.

Autocorrelation Plot

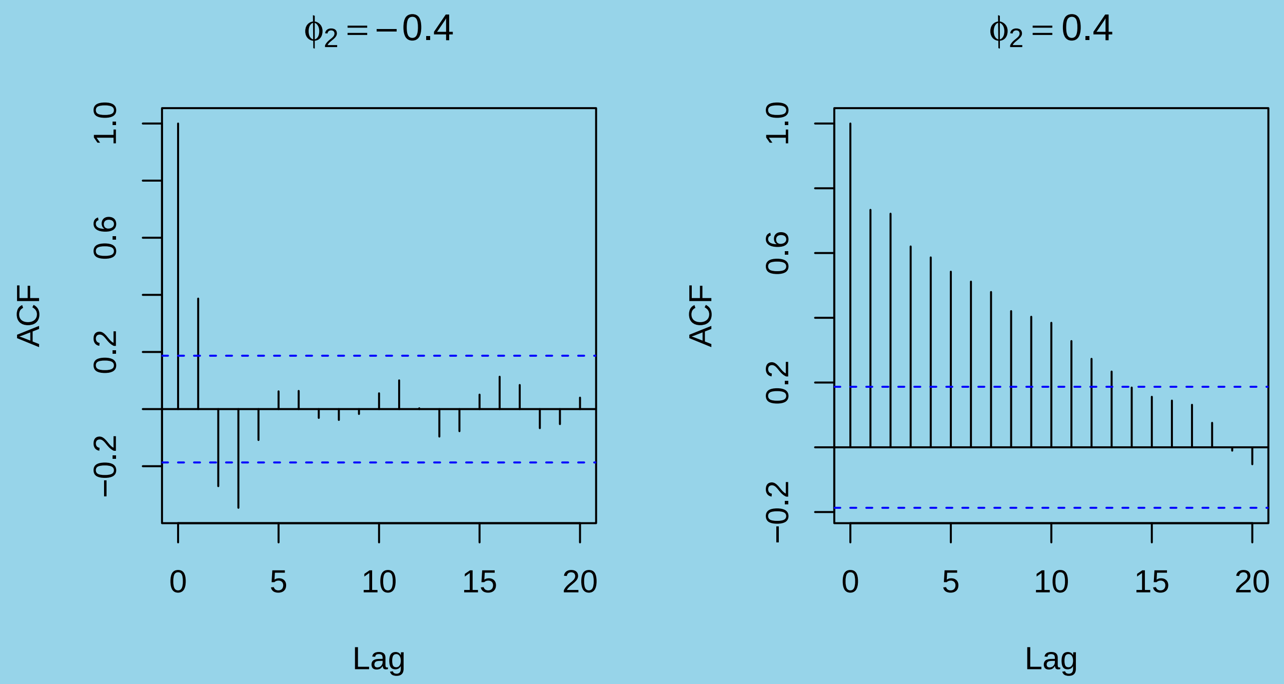


Sinusoidal pattern is not accidental!

The AR(2) model has a close relationship with the differential equation: $y'' = \beta_1 y' + \beta_2 y$, which can model a *harmonic oscillator* such as a pendulum.

ACF Plots for Simulated AR(2) Data

```
par(mfrow=c(1,2))
acf(arima.sim(110, model=list(ar=c(.5, -.4))),
    main=expression(phi[2]==-.4))
acf(arima.sim(110, model=list(ar=c(.5, .4))),
    main=expression(phi[2]==.4))
```



The first plot oscillates somewhat like the lynx data ACF.

Fitting an AR(2) Model to Data

We can fit the AR(2) model using maximum likelihood estimation for the parameters, using the arima() function:

```
arima(sqrt(lynx), order=c(2,0,0))

##
## Call:
## arima(x = sqrt(lynx), order = c(2, 0, 0))
##
## Coefficients:
##          ar1      ar2  intercept
##       1.3088  -0.7104    34.1280
## s.e.  0.0648   0.0645     2.0449
##
## sigma^2 estimated as 76.51:  log likelihood = -410.13,  aic = 828.26
```

Fitting an AR(2) Model to Data

The fitted model is

$$Z_n = 1.31Z_{n-1} - .7104Z_{n-2} + \varepsilon_n$$

where the error variance is estimated to be 76.51.

The mean level was estimated as 34.12, so $Z_n = \sqrt{Y_n} - 34.12$**, where** $Y_j =$ **# of lynx trapped in year** j**, for** $j = 1821, ..., 1934$

Using Simulation as a Model Check

We can use simulation to check if an AR(2) model is appropriate for given data.

Simulation with a `for()` **loop is possible, as it was for the AR(1) model, and the** `arima.sim()` **function can be used with two values for the** `ar` **parameter in place of one.**

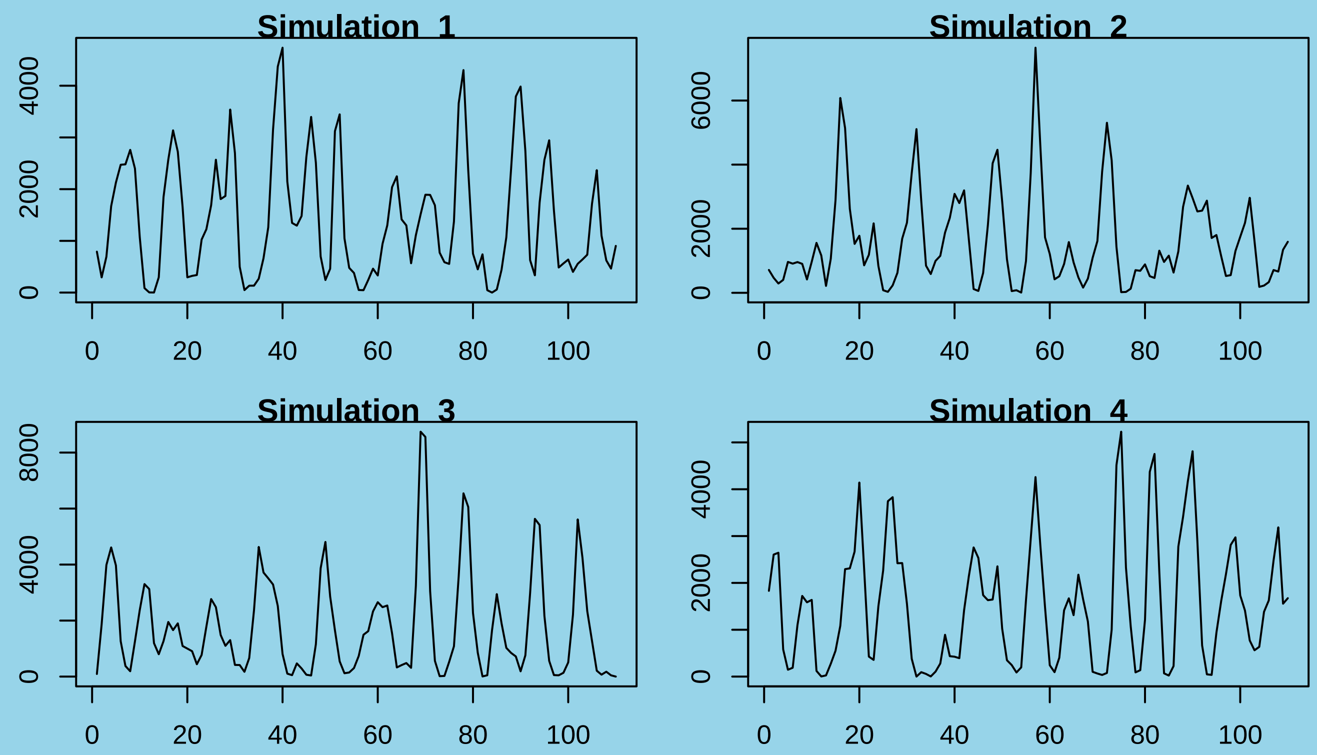
Simulating an AR(2) Model

The next figure shows the results of simulating from the AR(2) model for the lynx data.

Simulations from the fitted model are compared with real data.

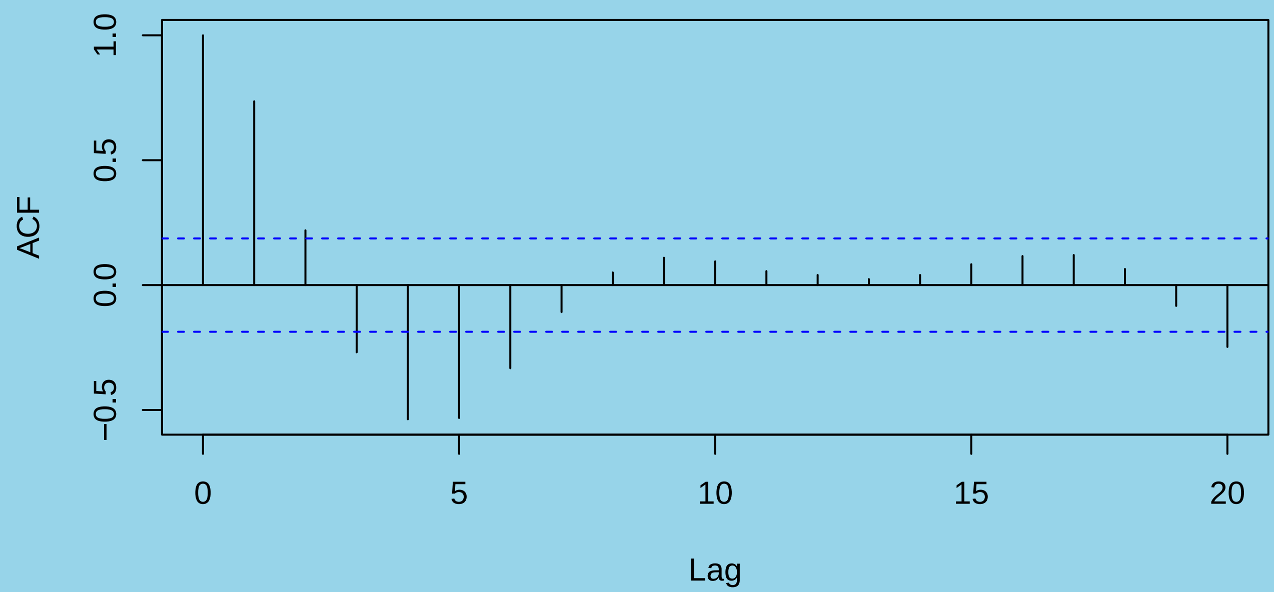
```
par(mfrow=c(2,2));  pars <- c(1.3088, -.7104)
sig <- sqrt(76.51); xbar <- 34.12
for (j in 1:4) {
ts.plot((arima.sim(110, model=list(ar=pars),
       sd=sig)+xbar)^2, ylab="", main=
       paste("Simulation ", j))
}
```

Simulating an AR(2) Model



This simulates the AR(2) process having the parameters that were estimated on the square root of the lynx data:

```
acf(arima.sim(110, model=list(ar=pars), sd=sig), main="")
```



Prediction from an AR(2) Model

The `predict.ARIMA()` function can be used to predict future values of this process as well:

```
lynx.ar2 <- arima(sqrt(lynx), order=c(2, 0, 0)) # store fitted model
predict(lynx.ar2, n.ahead=10) # predict 10 years into the future

## $pred
## Time Series:
## Start = 1935
## End = 1944
## Frequency = 1
##  [1] 53.36 42.14 30.96 24.28 23.50 27.21 32.62
##  [8] 37.07 39.05 38.48
##
## $se
## Time Series:
## Start = 1935
## End = 1944
## Frequency = 1
##  [1]  8.747 14.407 16.866 17.194 17.294 17.947
##  [7] 18.622 18.877 18.884 18.945
```

Prediction from an AR(2) Model

To predict on the original scale, we could get a rough approximate prediction by squaring the predictions:

```
predict(lynx.ar2, n.ahead=10)$pred^2

## Time Series:
## Start = 1935
## End = 1944
## Frequency = 1
##  [1] 2847.1 1776.0  958.3  589.7  552.1  740.2
##  [7] 1064.2 1374.5 1525.2 1480.8
```

More work would be needed to convert the standard errors - or a bootstrap procedure could be used.

Summary of Results and Model Limitations

The predictions into the future amount to a form of extrapolation so caution would be warranted when interpreting the results. Is lynx trapping operational in the same way as in past?

As an approximation, the AR(2) model is a good start, providing more realism than either independence or even an AR(1) model could provide.

However, the behaviour of the simulated data tends to be somewhat less regular than the real data, suggesting that an important factor might be missing from the analysis.

Moving Average Processes

Unlike AR processes where the data points are all dependent on each other, the data points of an MA process only depend on each other if they close together in time.

This offers new modelling possibilities.

The Moving Average Process of Order 1

The 0-mean moving average process of order 1 (MA(1)) is defined as

$$Z_n = \theta_1 \varepsilon_{n-1} + \varepsilon_n$$

where the ε's are independent and normally distributed with mean 0 and variance σ_2, and θ_1 is a constant.

According to this definition, any ε_k will be independent of Z_j for all $j < k$.

If we define $Y_n = Z_n + \mu$, then Y_n is a MA(1) process with mean μ.

Because the ε's have expectation 0, it follows that $E[Z_n] = 0$.

Is this a Markov process?

The MA(1) process is *not* a Markov process, since

$$Z_n = \varepsilon_n + \theta_1 \varepsilon_{n-1}$$

and

$$\varepsilon_{n-1} = Z_{n-1} - \theta_1 \varepsilon_{n-2}$$

so

$$Z_n = \varepsilon_n + \theta_1 Z_{n-1} - \theta_1^2 \varepsilon_{n-2}.$$

But

$$\varepsilon_{n-2} = Z_{n-2} - \theta_1 \varepsilon_{n-3}$$

so

$$Z_n = \varepsilon_n + \theta_1 Z_{n-1} - \theta_1^2 Z_{n-2} + \theta_1^3 \varepsilon_{n-3}.$$

And continuing, we would see that Z_n depends explicitly on all previous Z's. This means that prediction of Z_n, given Z_{n-1} could be improved upon by making use of earlier Z's, contradicting the Markov property.

Development of the Variance for the MA(1) Process

By squaring both sides of the defining equation and taking expectations, we have

$$E[Z_n^2] = \sigma^2(1 + \theta_1^2).$$

This is the variance of Z_n, because $E[Z_n] = 0$.

Details: Squaring the right hand side gives $\theta_1^2\varepsilon_{n-1}^2 + \varepsilon_n^2 + 2\theta_1\varepsilon_n\varepsilon_{n-1}$. Since the ε's are independent, $E[\varepsilon_n\varepsilon_{n-1}] = E[\varepsilon_n]E[\varepsilon_{n-1}] = 0$ so the expected value of the right hand side is $\sigma^2\theta_1^2 + \sigma^2$.

Development of the First Lag Autocovariance

By multiplying the defining equation by Z_{n-1} and taking expectations, show that

$$E[Z_n Z_{n-1}] = \theta_1 \sigma^2.$$

Details: To do this, you will need to use the fact that the defining equation also implies that

$$Z_{n-1} = \theta_1 \varepsilon_{n-2} + \varepsilon_{n-1}.$$

Then

$$Z_n Z_{n-1} = \varepsilon_n \varepsilon_{n-1} + \theta_1 \varepsilon_{n-1}^2 + \theta_1 \varepsilon_n \varepsilon_{n-2} + \theta_1^2 \varepsilon_{n-1} \varepsilon_{n-2}.$$

Taking expectations of this and using independence of the ε's gives

$$E[Z_n Z_{n-1}] = \theta_1 E[\varepsilon_{n-1}^2] = \theta_1 \sigma^2.$$

Thus, the covariance of Z_n and Z_{n-1} is $\theta_1 \sigma^2$. This is the first lag autocovariance.

Development of the First Lag Autocovariance

Since $[Z_n] = 0$, the covariance is just E[Z$_n Z_{n-1}$].

Use the same procedure as in the preceding part to show that

$$E[Z_n Z_{n-k}] = 0$$

for $k = 2, 3, \ldots$. We deduce that the covariance of all autocovariances at lags larger than 1 must be 0.

Details:

$$E[Z_n Z_{n-k}] = E[(\varepsilon_n + \theta_1 \varepsilon_{n-1})(\varepsilon_{n-k} + \theta_1 \varepsilon_{n-k-1})] = 0$$

because all of the products inside the expectation involve different ε's and, therefore, must have expectation 0, because of independence.

Development of the First Lag Autocorrelation

The kth autocorrelation is defined as the kth autocovariance divided by the variance of the process. We deduce that all autocorrelations beyond lag 1 are 0, and that the first one is nonzero.

Details: Since the autocovariances beyond lag 1 are all 0, dividing by anything still gives 0. The lag 1 autocovariance is nonzero so dividing by anything still gives a nonzero result.

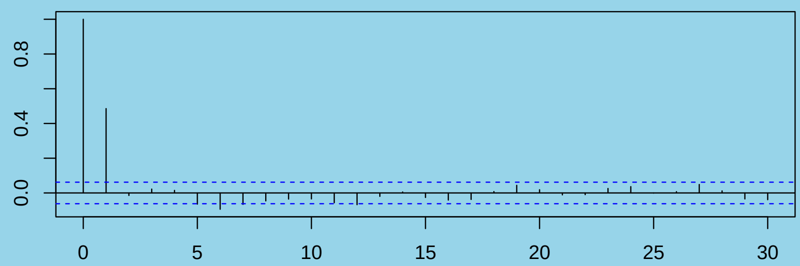
Simulation of MA(1) Data

Simulate 1000 observations from an MA(1) process with $\theta_1 = 0.9$, using the following code

```
ma1 <- arima.sim(1000, model=list(ma=c(.9)))
```

The sample autocorrelation function for the data is obtained as follows:

```
acf(ma1)
```



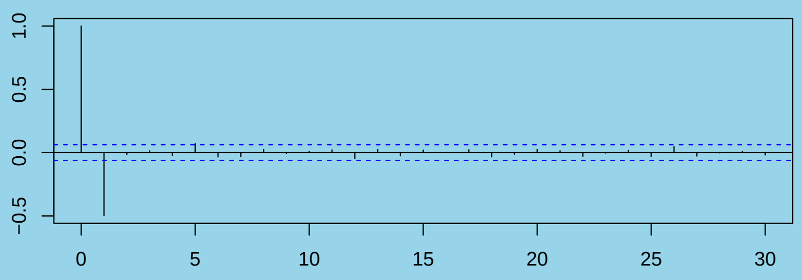
The autocorrelations are all 0 beyond lag 1 as expected, and the first one is positive which agrees with the formula.

Simulating an MA(1) Process with a Negative Parameter

Repeating the preceding simulation but with $\theta_1 = -0.9$:

```
ma1 <- arima.sim(1000, model=list(ma=c(-.9)))
```

```
acf(ma1)
```



The autocorrelations are all 0 beyond lag 1 as expected and the first one is negative which agrees with the formula.

Distinguishing Between MA(1) and AR(1) Processes

Given a set of time series data where you are faced with the question as to whether an AR(1) or MA(1) process is appropriate as a model, what action would you take, and how would you make your choice?

Answer: Plot the sample acf function and check to see if the autocorrelations cut off after lag 1 (indicating MA(1)) or if they decay exponentially (indicating AR(1)).

Fitting an MA Process to Data

The `arima()` function can be used to fit a moving average model to data.

For the data simulated earlier, in the vector `ma1`, we would use

```
ma1.fit <- arima(ma1, order=c(0, 0, 1))  # the third
# component of order is the MA order
ma1.fit

##
## Call:
## arima(x = ma1, order = c(0, 0, 1))
##
## Coefficients:
##           ma1  intercept
##        -0.897      0.002
## s.e.    0.014      0.003
##
## sigma^2 estimated as 1.02:  log likelihood = -1431,  aic = 2868
```

Fitting an MA Process to Data

The fitted model is

$$\widehat{Y}_n = \widehat{\mu} + \theta_1 \widehat{\varepsilon}_{n-1}.$$

where ε_{n-1} is the difference (residual) between Y_{n-1} and $\widehat{Y}_{n-1}$. ($\widehat{Y}_0 = \widehat{\mu}$.) Plugging in the results from the output, we have

$$\widehat{Y}_n = -.8946 \widehat{\varepsilon}_{n-1}.$$

Making Predictions

Predicting the next 5 observations can be done with the predict.ARIMA function as follows:

```
predict(ma1.fit, n.ahead=5)

## $pred
## Time Series:
## Start = 1001
## End = 1005
## Frequency = 1
## [1] -0.649028  0.002103  0.002103  0.002103
## [5]  0.002103
##
## $se
## Time Series:
## Start = 1001
## End = 1005
## Frequency = 1
## [1] 1.012 1.359 1.359 1.359 1.359
```

The Moving Average Process of Order 2 (MA(2))

This is defined as

$$Z_n = \theta_2 \varepsilon_{n-2} + \theta_1 \varepsilon_{n-1} + \varepsilon_n$$

where the ε's are independent and normally distributed with mean 0 and variance σ_2, θ_2 and θ_1 are constant.

According to this definition, any ε_k will be independent of Z_j for all $j < k$.

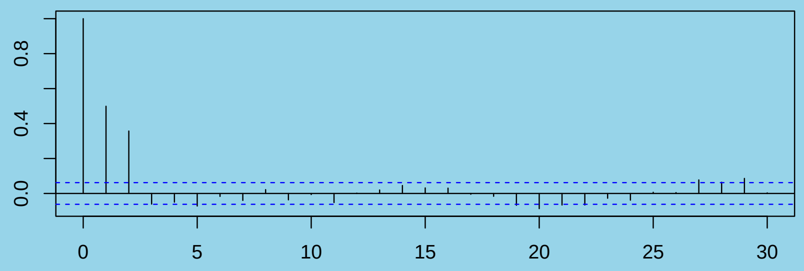
If we define $Y_n = Z_n + \mu$, then Y_n is a MA(2) process with mean μ.

Simulation of MA(2) Data

We can simulate 1000 observations from an MA(2) process with $\theta_1 = 0.5$ and $\theta_2 = 0.7$ using the following code

```
ma2 <- arima.sim(1000, model=list(ma=c(.5, .7)))
```

```
acf(ma2)
```



The autocorrelations are all 0 beyond lag 2 as expected and the first two are nonzero.

The ACF gives us a way to distinguish data following an MA(2) process from an MA(1) process.

Fitting MA(2) Models

Use the `arima()` function to estimate the parameters for given data.

For the `ma2` data, we would use

```
ma2.ma2 <-  arima(ma2, order=c(0, 0, 2)) # Order MA(2)
```

Predicting Future Values in MA(2) Models

We use the `predict.Arima()` function to predict the next values in the sequence. To predict the next 10 values of the `ma2` series, we use

```
predict(ma2.ma2, n.ahead=10)

## $pred
## Time Series:
## Start = 1001
## End = 1010
## Frequency = 1
##  [1] -0.96651 -1.32366 -0.06941 -0.06941 -0.06941
##  [6] -0.06941 -0.06941 -0.06941 -0.06941 -0.06941
##
## $se
## Time Series:
## Start = 1001
## End = 1010
## Frequency = 1
##  [1] 1.005 1.140 1.323 1.323 1.323 1.323 1.323
##  [8] 1.323 1.323 1.323
```

The autoregressive-moving average process of order 1, 1 (ARMA(1, 1))

This is defined as

$$Z_n = \phi_1 Z_{n-1} + \theta_1 \varepsilon_{n-1} + \varepsilon_n$$

where the ε's are independent and normally distributed with mean 0 and variance σ_2, θ_2 and θ_1 are constant.

According to this definition, any ε_k will be independent of Z_j for all $j < k$.

If we define $Y_n = Z_n + \mu$, then Y_n is an ARMA(1) process with mean μ.

Simulating ARMA(1,1) Data

Simulate 1000 observations from an ARMA(1, 1) process with $\phi_1 = 0.5$ and $\theta_2 = 0.7$ using the following code

```
arma11 <- arima.sim(1000, model=list(ar=c(.5), ma=c(.7)))
```

Fitting and Predicting with ARMA(1,1) Models

Use the `arima()` function to estimate the parameters.

For the `arma11` data set, we would use

```
arma11.arma11 <- arima(arma11, order=c(1, 0, 1))
```

Use the `predict.Arima()` function to predict the next values in the series.

The Integrated Autoregressive-Moving Average Process (ARIMA)

ARIMA processes allow for random trends.

The ARIMA(1, 1, 1) is defined as

$$X_n = X_{n-1} + Z_n$$

where

$$Z_n = \phi_1 Z_{n-1} + \theta_1 \varepsilon_{n-1} + \varepsilon_n$$

and where the ε's are independent and normally distributed with mean 0 and variance σ_2, θ_2 and θ_1 are constant.

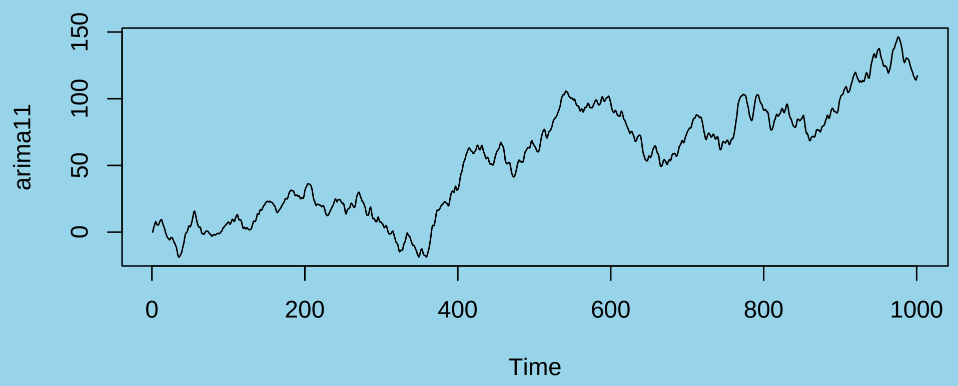
An ARIMA(1,1,1) process with drift μ is defined as $X_n = X_{n-1} + Y_n$ where $Y_n = Z_n + \mu$; this incorporates a deterministic trend as well as a random trend.

Simulating ARIMA Data

We can simulate 1000 observations from an ARMA(1, 1, 1) process with $\phi_1 = 0.5$ and $\theta_2 = 0.7$ using the following code

```
arima11 <- arima.sim(1000, model=list(order=c(1, 1, 1),
      ar=c(.5), ma=c(.7)))
```

```
ts.plot(arima11)
```



Note the apparent trend. This is not systematic and could just as likely trend downward.

Fitting and Predicting with ARIMA Models

Use the `arima()` function to estimate the parameters.

For the simulated data in `arima11`, we would use

```
arima11.arima <- arima(arima11, order=c(1,1,1))
# the middle one indicates 1 integration order
```

Use the `predict.Arima()` function to predict the next values in the series.

Changing the Noise Standard Deviation

In order to simulate data with a different noise standard deviation, use the `sd` argument in the `arima.sim()` function as, for example, with $\sigma = 10$:

```
arima11 <- arima.sim(1000, model=list(order=c(1, 1, 1),
    ar=c(.5), ma=c(.7)), sd=10)
```

Automatic Fitting of ARIMA Models Using AIC

The `auto.arima()` function in the *forecast* package uses AIC (and related criteria) to automatically choose from among the different models. Models with smaller AIC values are preferred.

The AIC criterion balances the goodness of fit of the model to the data via maximum likelihood estimation with a penalty on the complexity of the model.

In other words, a model with many parameters, that is, a very complex model, might fit the data very well, while a model with only a few parameters is simple but might not fit the data well.

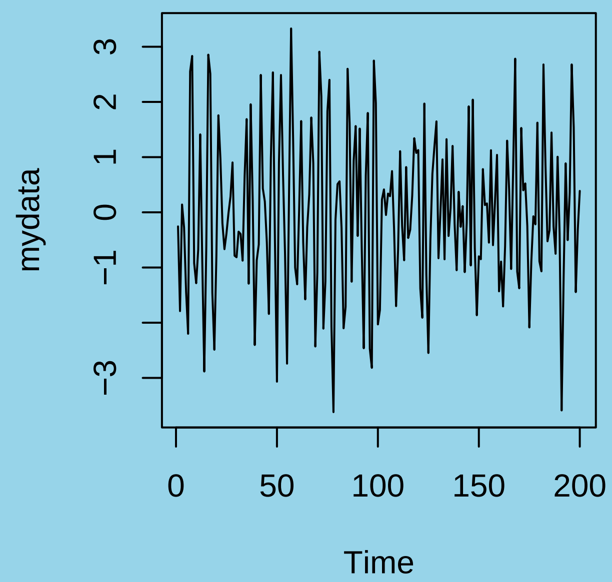
AIC strikes a balance between these simplicity and goodness of fit.

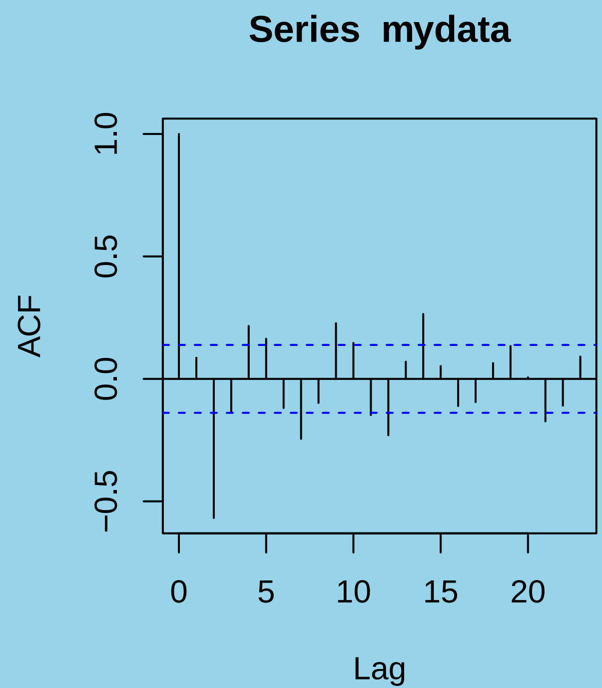
Some Illustrative Examples

```
mydata <- arima.sim(200, model=list(ma=c(.1, -.85))) # MA(2) data
library(forecast)
auto.arima(mydata)

## Series: mydata
## ARIMA(2,0,0) with zero mean
##
## Coefficients:
##          ar1      ar2
##        0.136  -0.578
## s.e.   0.057   0.057
##
## sigma^2 estimated as 1.35:  log likelihood=-313.1
## AIC=632.2   AICc=632.4   BIC=642.1
```

Plotting Simulated MA(2) Data

```r
par(mfrow=c(1,2))
ts.plot(mydata); acf(mydata)
```



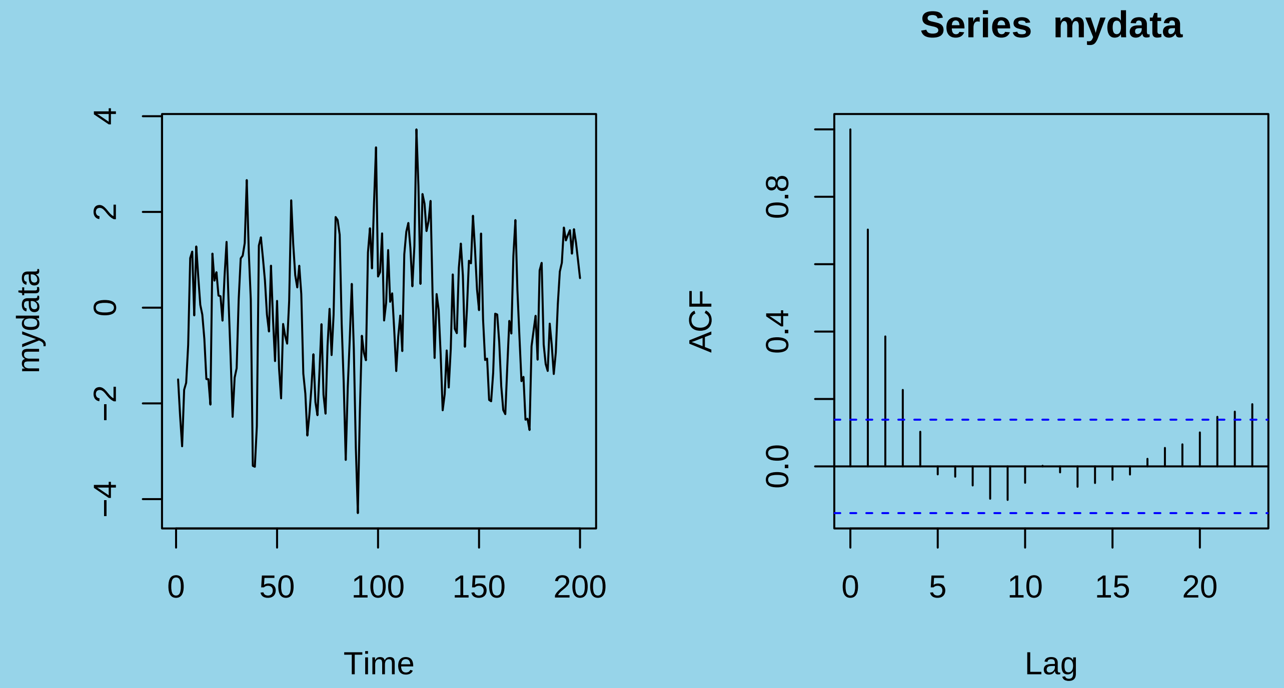


Simulating and Fitting ARMA(1,1) Data

```
mydata <- arima.sim(200, model=list(ar=c(.6), ma=c(.3))) # ARMA(1,1)
auto.arima(mydata)

## Series: mydata
## ARIMA(1,0,1) with zero mean
##
## Coefficients:
##          ar1     ma1
##        0.544   0.342
## s.e.   0.078   0.086
##
## sigma^2 estimated as 0.967:  log likelihood=-279.9
## AIC=565.7   AICc=565.8   BIC=575.6
```

Plotting Simulated ARMA(1,1) Data

```
par(mfrow=c(1,2))
ts.plot(mydata); acf(mydata)
```

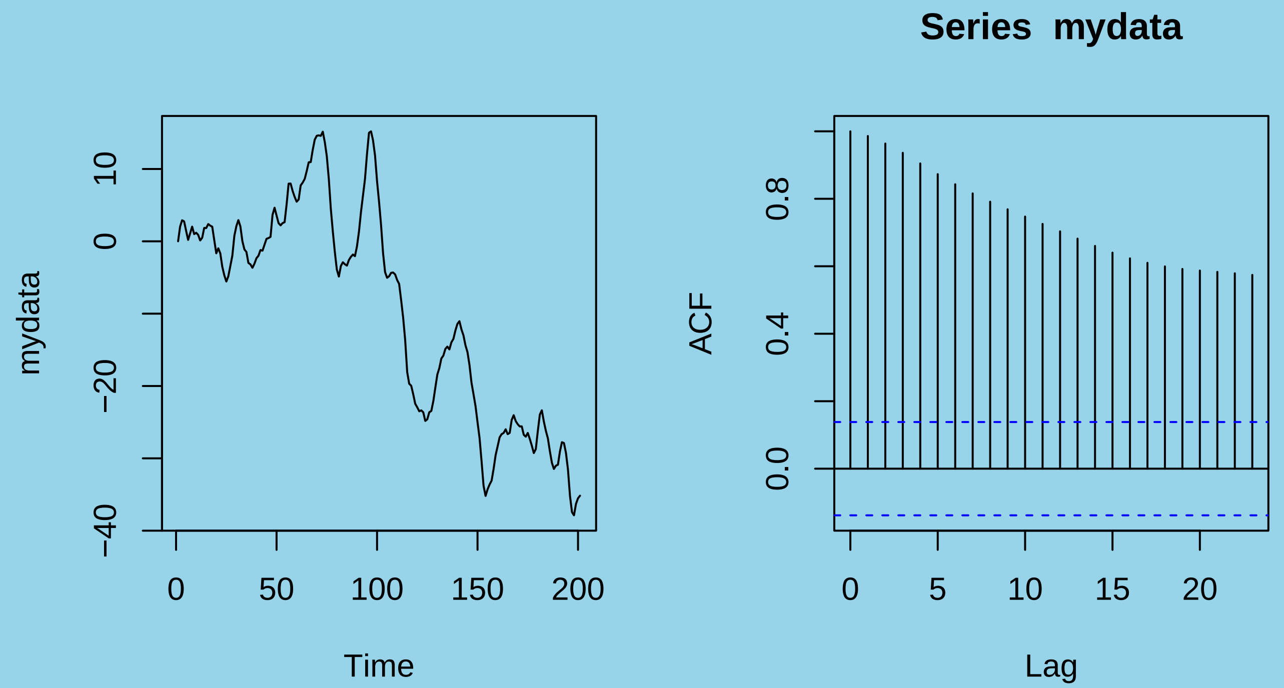


Simulating and Fitting ARIMA(1,1,1) Data

```
mydata <- arima.sim(200, model=list(order=c(1,1,1), ar=c(.6), ma=c(.3)))
auto.arima(mydata)

## Series: mydata
## ARIMA(1,1,1)
##
## Coefficients:
##          ar1    ma1
##        0.586  0.351
## s.e.  0.076  0.093
##
## sigma^2 estimated as 0.99:  log likelihood=-282.3
## AIC=570.5   AICc=570.7   BIC=580.4
```

Plotting Simulated ARIMA(1,1,1) Data

```
par(mfrow=c(1,2))
ts.plot(mydata); acf(mydata)
```

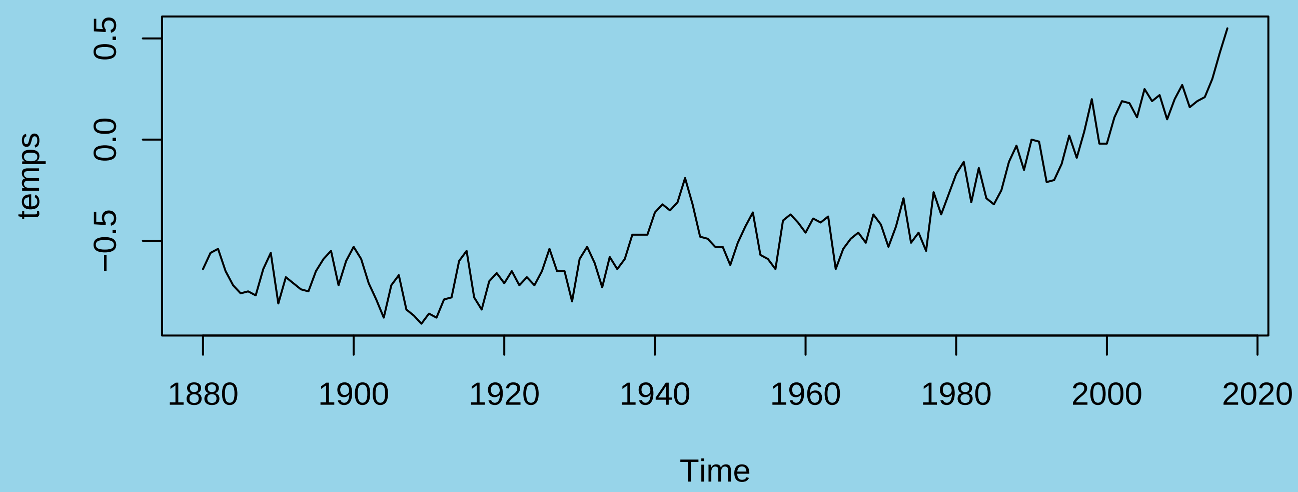


The ACF has no meaning here, since this process is not stationary - it has random trends as seen on the plot on the left.

Application to Global Warming

The data in `Globaltemps.R` are the differences in the global average temperature from that of 1990 for the years 1880 through 2016.

```
source("Globaltemps.R")
temps <- ts(temps, start = 1880, end = 2016)
ts.plot(temps)
```



Data are from Datahub (https://datahub.io/core/global-temp).

Fitting an ARIMA(1,1,1) Model

```
temps.arima <- arima(temps, order = c(1,1,1))
temps.arima

##
## Call:
## arima(x = temps, order = c(1, 1, 1))
##
## Coefficients:
##          ar1      ma1
##        0.352   -0.702
## s.e.   0.144    0.104
##
## sigma^2 estimated as 0.0104:  log likelihood = 117.7,  aic = -229.3
```

This assures us that there is nonstationarity in the data - of the kind that has probably always been operating for millenia - i.e. random trends.

Is there a Deterministic Trend?

A deterministic trend in a nonstationary model is called `drift`.

We can informally check to see if there are trends with slope .001, .003, .005, .007 or .009 in the 137 observations by subtracting such trends out and comparing the resulting AIC values. (Remember we want to minimize AIC.)

Is there a Deterministic Trend?

```
temps.arima <- arima(I(temps-.001*(1:137)), order = c(1,1,1))
temps.arima$aic

## [1] -230.3

temps.arima <- arima(I(temps-.003*(1:137)), order = c(1,1,1))
temps.arima$aic

## [1] -231.9

temps.arima <- arima(I(temps-.005*(1:137)), order = c(1,1,1))
temps.arima$aic

## [1] -233.2

temps.arima <- arima(I(temps-.007*(1:137)), order = c(1,1,1))
temps.arima$aic

## [1] -233.8

temps.arima <- arima(I(temps-.009*(1:137)), order = c(1,1,1))
temps.arima$aic

## [1] -233.7
```

It looks like there might be drift value of about .007. (You could also use `auto.arima()` **to get a similar result.)**

Using Simulation to Compare Scenarios

Our fitted model output:

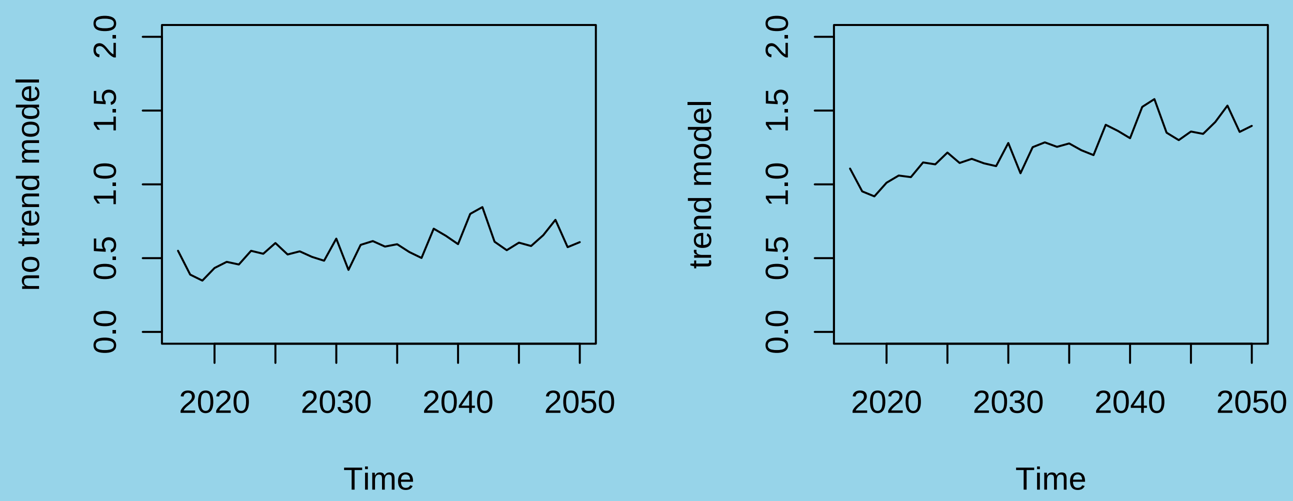
```
temps.arima <- arima(I(temps-.007*(1:137)), order = c(1,1,1))
temps.arima

##
## Call:
## arima(x = I(temps - 0.007 * (1:137)), order = c(1, 1, 1))
##
## Coefficients:
##          ar1      ma1
##        0.394   -0.775
## s.e.   0.131    0.088
##
## sigma^2 estimated as 0.01:  log likelihood = 119.9,  aic = -233.8
```

Comparing No Trend Scenario at 2050 with Trend Scenario

```r
n <- 34 # forecast to 2050 from 2016
temp.sim <- arima.sim(n-1, model=list(order=c(1,1,1),
    ar=c(.394), ma = -.775), sd = sqrt(.01)) + temps[137]
temptrend.sim <- temp.sim +.007*(1:n) + temps[137]
notrendsim <- ts(temp.sim, start=2017,
     end=2017+n-1) # convert to ts object
trendsim <- ts(temptrend.sim, start=2017,
     end=2017 + n-1) # convert to ts
```

Single Simulation Realizations from the Two Possible Scenarios

```
par(mfrow=c(1,2))
ts.plot(notrendsim, ylim=c(0,2), ylab="no trend model")
ts.plot(trendsim, ylim=c(0,2), ylab="trend model")
```



Projected Temperature Increase by 2050

Repeated simulations under the two scenarios allows us to make a comparison at any percentile level we wish, such 2.5%, 50% and 97.5%, as here:

```
Nsims <- 10000
temp2050 <- numeric(Nsims)
for (j in 1:Nsims) {
    temp.sim <- arima.sim(n-1, model=list(order=c(1, 1,1),
         ar=c(.394), ma = -.775), sd = sqrt(.01)) + temps[137]
    temp2050[j] <- temp.sim[n]
}
quantile(temp2050, c(.025, .5, .975)) # no trend

##     2.5%      50%    97.5%
## 0.06429 0.54778 1.04056

quantile(temp2050+.007*n, c(.025, .5, .975))  # with trend

##   2.5%    50%  97.5%
## 0.3023 0.7858 1.2786
```

These are empirical projections and do not incorporate any of the science behind global circulation models.

Projected Temperature Increases by 2100

The same kind of simulation exercise can be carried out for projections to 2100:

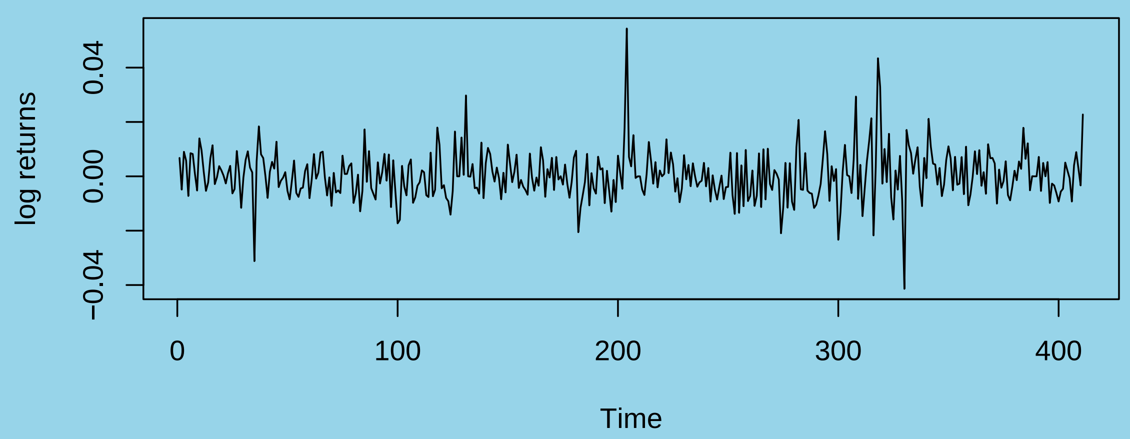
```
temp2100 <- numeric(Nsims)
n <- 84 # number of years from 2016 to 2100
for (j in 1:Nsims) {
    temp.sim <- arima.sim(n-1, model=list(order=c(1,1,1),
         ar=.394, ma = -.775), sd = sqrt(.01)) + temps[137]
    temp2100[j] <- temp.sim[n]
}
quantile(temp2100, c(.025, .5, .975)) # no trend

##    2.5%     50%   97.5%
## -0.1671  0.5447  1.2453

quantile(temp2100+.007*n, c(.025, .5, .975))  # with trend

##   2.5%    50%  97.5%
## 0.4209 1.1327 1.8333
```

Another Markov Process - ARCH model

The ARCH model, and its more sophisticated variant, GARCH, are important models used to analyze financial time series, such as stock indices and treasury bond yields.

The following trace plot shows daily log returns for the FTSE (Financial Times Stock Exchange) for 1991 and 1992:

```
logreturns <- diff(log(EuStockMarkets[1:412, 4]))
ts.plot(logreturns, ylab="log returns")
```

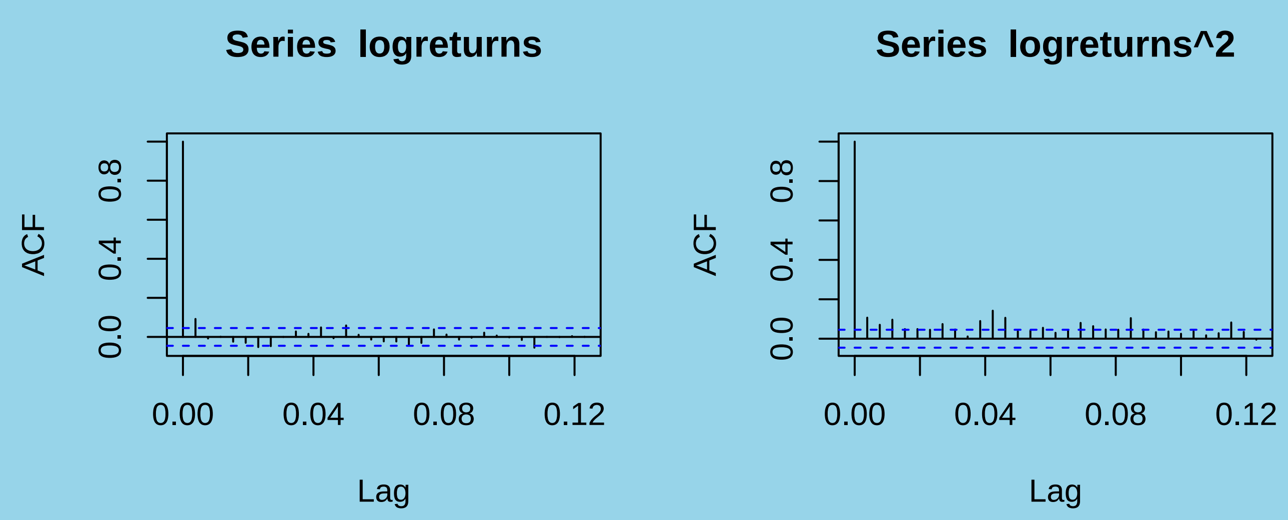


The log return for day t is the logarithm of x_t/x_{t-1}. It gives an indication as to how well the market is doing. A positive log return means that the market went up.

Analyze the full data set:

```
logreturns <- diff(log(EuStockMarkets[, 4]))
```

```
par(mfrow=c(1,2))
acf(logreturns)
acf(logreturns^2)
```



There is a slight autocorrelation at lag 1 in the raw log returns, but there is somewhat more autocorrelation in the squared log returns (right panel).

The ARCH model

A model which gives similar behaviour to the daily log returns for the FTSE is the following:

For day t, the log return is given by

$$y_t = s_t Z_t$$

where

$$s_t = \sqrt{a_0 + a_1 y_{t-1}^2 + a_2 y_{t-2}^2 + a_3 y_{t-3}^2}$$

and Z_t is a standard normal random variable.

The parameter values are $a_0 = 0.00001$, $a_1 = 0.1$ and $a_2 = 0.05$ and $a_3 = .08$. Note this model is a lot like an autoregressive process of order 3 in terms of y_t^2.

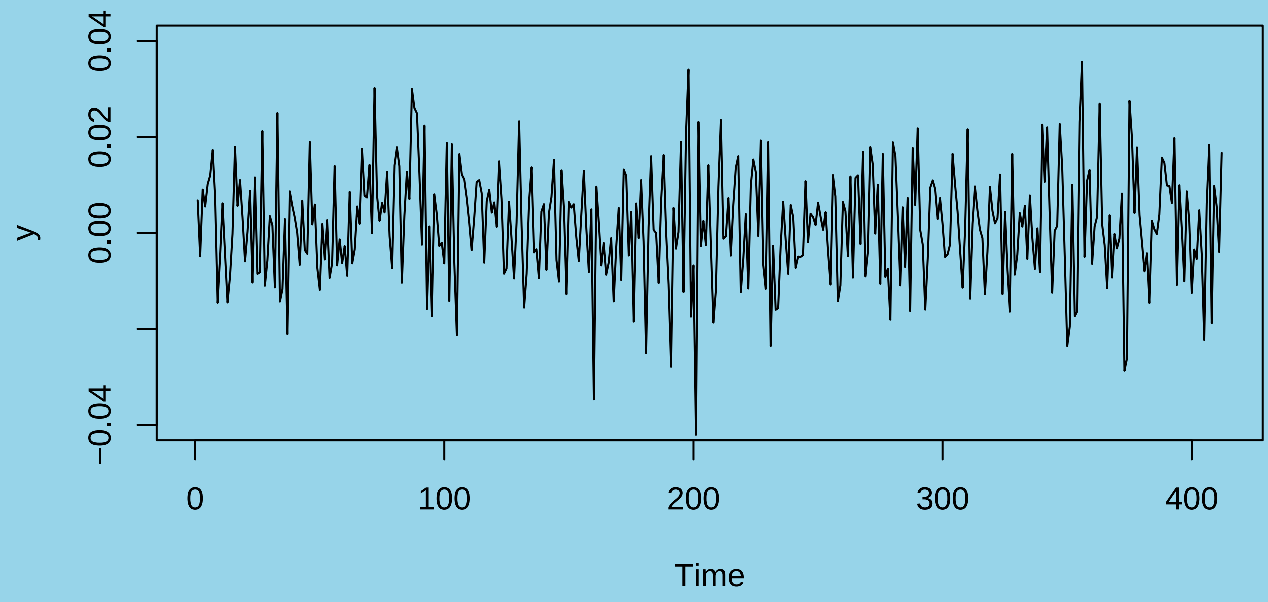
We can start of a simulation by setting the first three values of y_t to the first three values from the observed log returns.

The ARCH model

```
n <- 412 # number of days for the simulation
a <- c(.1, .05, .08); a0 <- .0001
y <- numeric(n); y[1:3] <- logreturns[1:3]
for (t in 4:n){
    s <- sqrt(a0 + a[1]*y[t-1]^2 + a[2]*y[t-2]^2 +
        a[3]*y[t-3]^2)
    y[t] <- s*rnorm(1)
}
```

The ARCH Model

```
ts.plot(y, ylim=c(-.04, .04))
```



As in the actual series, the simulated values hover between $\pm.01$ but occasionally between $\pm.04$.

We might use this model to predict future behaviour of the FTSE, such as how long it might take to exceed some value, such as .04.

The ARCH Model

Suppose we want to simulate the ARCH process until the first time it exceeds 0.04.

We don't know beforehand when this will occur, we wouldn't know how to stop the `for()` loop at the right time, so we should use a `while()` loop.

The ARCH Model

```
a <- c(.1, .05, .08); a0 <- .0001
y <- numeric(n); y[1:3] <- logreturns[1:3]
while (y[t] < .04){
    t <- t+1
    s <- sqrt(a0 + a[1]*y[t-1]^2 + a[2]*y[t-2]^2
        +a[3]*y[t-3]^2)
    y[t] <- s*rnorm(1)
}
print(t)

## [1] 1326
```

By repeatedly running this simulation, we could obtain a distribution of the times until we would expect the log returns to first exceed .04. This kind of information would be useful, for example, in pricing certain options.