Maximum Likelihood Estimation

Kushal Kharel

10/9/2021

Maximum Likelihood Estimation

One of the key problem in statistical inference is to find a way to infer from sample data to population parameters. MLE provides one solution to such problems. Consider a set of observations, $y_1, y_2, ..., y_n$ (can be time series or not), the likelihood function L is the joint probability density of obtaining the data that are actually observed. It is considered as a function of unknown parameters with the observed data held fixed. The maximum likelihood estimators are those values of the parameters for which the data actually observed are most likely that is the values that maximizes the likelihood function.

We have independent and identically distributed sample of data:

$$y_1, y_2, ..., y_n$$

We are assuming that the data is generated from some parametric density function. Let the data be generated from a known density function (or the probability mass function) which is indexed by the parameter θ.

$$y_i \sim f(y|\theta)$$

Then we can construct a likelihood function using the assumption that the data are iid which is simply the product of density or probability mass function evaluated at each y_i. We take the log of that function because it is computationally easier among other reasons.

$$L(\theta) = \prod_{i=1}^{N} f(y_i|\theta) \log(L(\theta)) = \sum_{i=1}^{N} \log f(y_i|\theta)$$

We may have infinite number of possible parameter values, and we want to choose the one that best fits the data. In order to do this, we need to maximize the log likelihood function defined above. Note that there may be some issues while carrying out the calculation. Suppose we assume the wrong density or probability mass function (mis-specification). Suppose that a unique maximizer does not actually exist (Identification).

Let us look at the linear model example defined below:

$$Y_i = \beta X_i + Z_i$$
$$Z_i = Y_i - \beta X_i$$
$$Z_i \sim N(0, \sigma^2)$$

$$Y_i \sim N(\beta X_i, \sigma^2)$$

Given that we know the density of Y_i which is defined below, we can construct a likelihood function which is the product of all the densities that are evaluated for each Y_i.

$$f(Y_i|\sigma^2,\beta) \sim \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{Y_i - \beta X_i}{2\sigma^2}\Big)$$

Then the likelihood and log likelihood function is given by;

$$L(\sigma^2,\beta) = \prod_{i=1}^{N} f(Y_i|\sigma^2,\beta) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{Y_i - \beta X_i}{2\sigma^2}\Big)$$

$$\log L(\sigma^2,\beta) = \sum_{i=1}^{n} -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}(Y_i - \beta X_i)^2$$

```
library(pacman)
```

```
## Warning: package 'pacman' was built under R version 4.0.5
```

```
p_load(data.table,fixest,lattice,magrittr,ggplot2)
```

Step 1: Generate the data

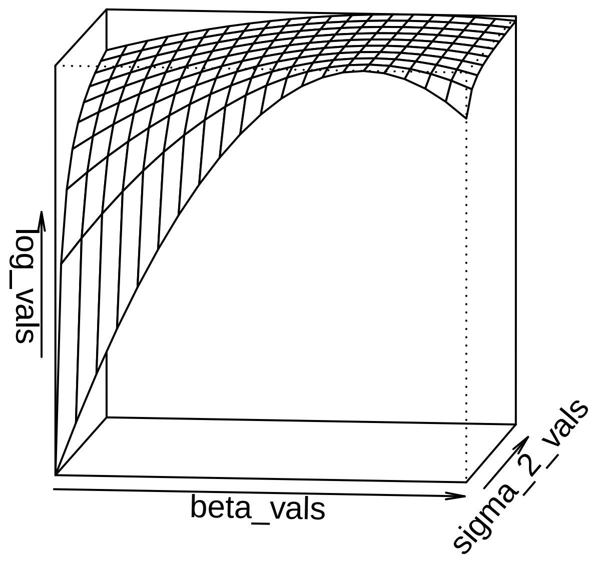
```
N = 500 # Sample size
beta = 5 # Beta
sigma_2 = 5 # sigma^2 (Distribution of Z)
DT = data.table(X = rnorm(N,0,5),
                Z = rnorm(N,0,sqrt(sigma_2))) %>%
  .[, Y := beta*X + Z] # Generating the dataset
```

Step 2: Derive the log likelihood function

```
# Likelihood function

log_like = function(theta,Y,X){ # theta is a vector where first component is beta and second component
  X = as.matrix(X); Y = as.matrix(Y); # converting X & Y into a matrix so we can vectorize the data
  N = nrow(X) # Set the sample size
  beta = theta[1] # Extract beta from its respective spot in a vector
  sigma_2 = theta[2] # Extract sigma from its respective spot in a vector
  e = Y - beta*X # Generating residuals
  loglik = -0.5*N*log(2*pi)-0.5*N*log(sigma_2) - ((t(e) %*% e)/(2*sigma_2)) # computing log-likelihood
  return(-loglik) # we are taking -loglik because the R built in optim function minimizes the functions
}
```

Let us plot the log likelihood function.

```r
# Plot the likelihood function
log_like_plot = function(beta,sigma_2){
  X = as.matrix(DT$X); Y = as.matrix(DT$Y);
  N = nrow(X)
  e = Y - beta*X
  loglik = -0.5*N*log(2*pi)-0.5*N*log(sigma_2) - ((t(e) %*% e)/(2*sigma_2))
  return(loglik)
}
log_like_plot = Vectorize(log_like_plot)

# Setting grid of beta and sigma_2 values
beta_vals = seq(-10,10,by=1)
sigma_2_vals = seq(1,10,by=1)
log_vals = outer(beta_vals, sigma_2_vals,log_like_plot)
persp(beta_vals, sigma_2_vals,log_vals,theta=7,phi=8,r=500)
```



The surface in the plot summarizes all the log likelihood values for different beta's and sigmas and what we want to do is find the values of beta and sigma squared that hits the highest part of the surface. We want to get different combination of beta and sigma squared values that maximizes the log likelihood function.

Step 3: Find MLE estimates

Now, in this step we are finding the values of beta and sigma squared that maximizes the log likelihood function or minimizes the negative log likelihood function.

```
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 4.0.3
```

```
MLE_estimates = optim(fn=log_like, # Likelihood function
                      par = c(1,1), # Initial guess of parameter values
                      lower = c(-Inf,-Inf), # Lower bound on parameters
                      upper = c(Inf,Inf), # Upper bound on parameters
                      hessian =TRUE, # Return Hessian matrix which we need to compute SE's
                      method = "L-BFGS-B",
                      # Custom Inputs that goes into out log_likelihood function
                      Y = DT$Y,
                      X = DT$X)
# Examine estimates
MLE_par = MLE_estimates$par # Extracting the value that minimizes the negative of log-likelihood functi
MLE_SE = sqrt(diag(solve(MLE_estimates$hessian))) # Extracting hessian by taking inverse of it and taki
MLE = data.table(param = c("beta", "sigma_2"),
                 estimates = MLE_par,
                 sd = MLE_SE) # combining them into a data table
kable(MLE)
```

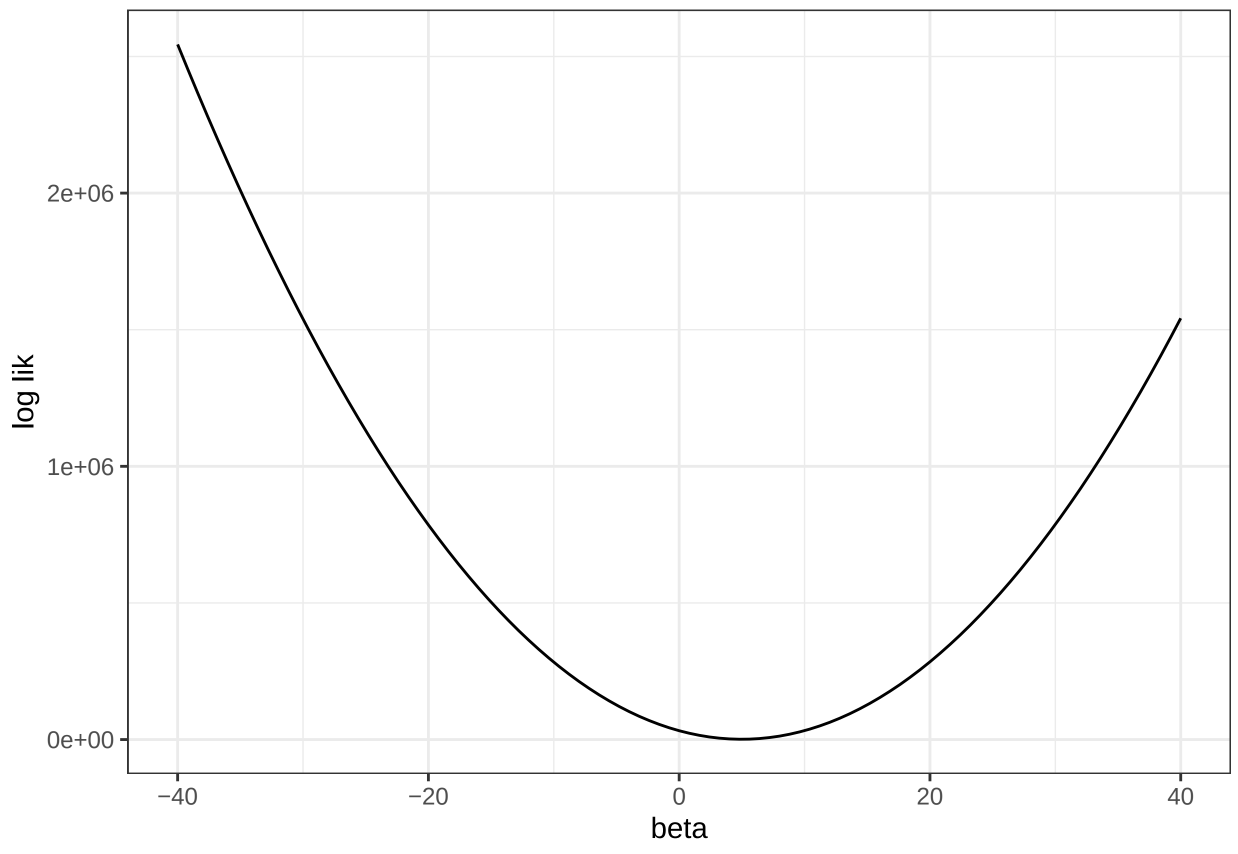
param	estimates	sd
beta	4.984993	0.0195415
sigma_2	4.798906	0.3035094

We can see that the maximum likelihood estimates for beta is 5.03 which is close to the true value of beta i.e., 5 and sigma squared is pretty far away and standard deviation is also quite large

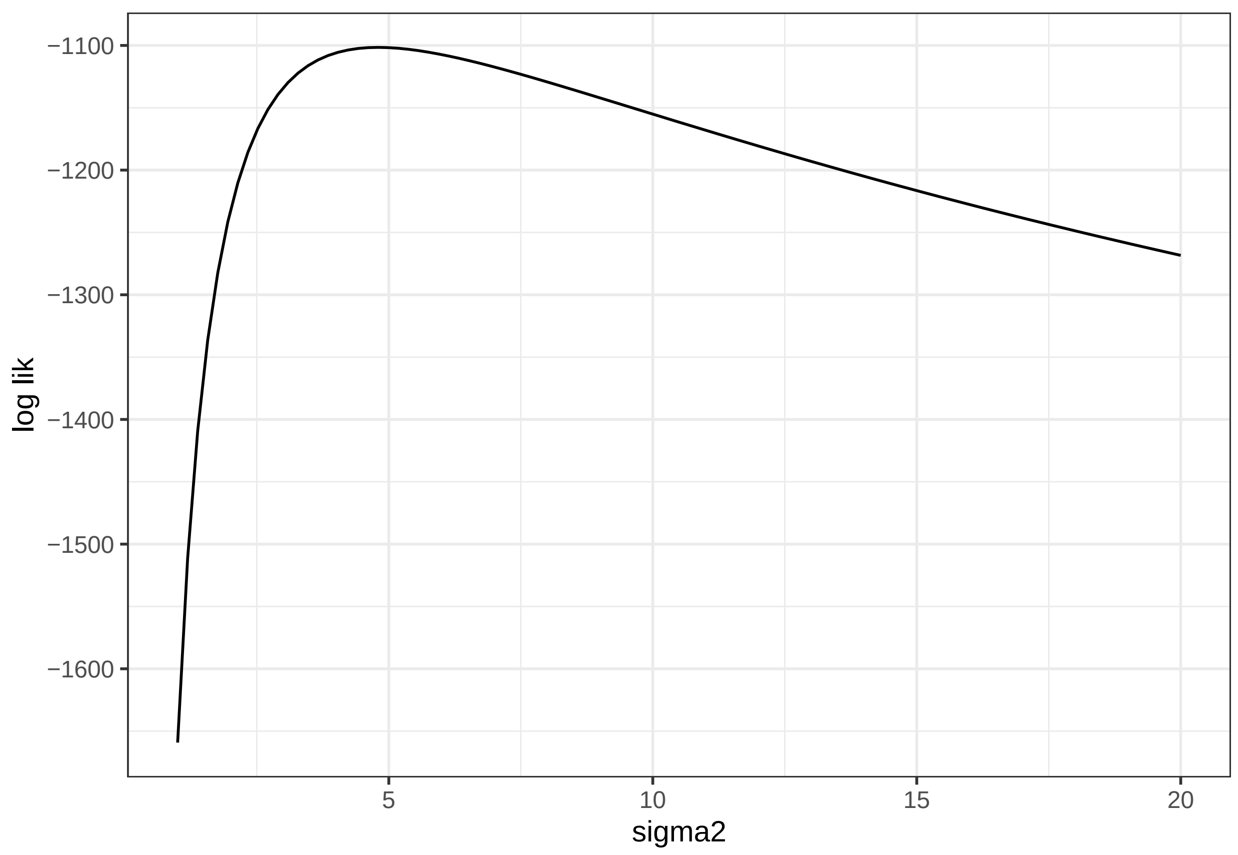
Step 4: Plot Estimates

```
log_like_plot_beta = function(beta){
  simga_2 = MLE_par[2]
  X = as.matrix(DT$X)
  Y = as.matrix(DT$Y)
  N = nrow(X)
  e = Y - beta*X
  loglik = -0.5*N*log(2*pi)-0.5*N*log(sigma_2) - ((t(e) %*% e)/(2*sigma_2))
  return(-loglik)
}

log_like_plot_sigma2 <- function(sigma_2){
  beta = MLE_par[1]
  X = as.matrix(DT$X)
  Y = as.matrix(DT$Y)
  N = nrow(X)
  e = Y - beta*X
  loglik  <- -.5*N*log(2*pi)-.5*N*log(sigma_2) - ( (t(e) %*% e)/ (2*sigma_2) )
  return(loglik)
```

```r
}

# Vectorize
log_like_plot_beta <- Vectorize(log_like_plot_beta)
log_like_plot_sigma2 <- Vectorize(log_like_plot_sigma2)

# Plots
beta <- ggplot(data = data.frame(beta = 0), mapping = aes(beta = beta)) +
    stat_function(fun = log_like_plot_beta) +
    xlim(-40,40) + theme_bw() +xlab("beta") + ylab("log lik")

sigma2 <- ggplot(data = data.frame(sigma2 = 0), mapping = aes(sigma2 = sigma2)) +
    stat_function(fun = log_like_plot_sigma2) +
    xlim(1,20) + theme_bw() +xlab("sigma2") + ylab("log lik")

beta
```



```r
sigma2
```



Binomial Model

Suppose we have the binomial distribution for N independent Bernoulli trials (e.g., coin flips, success or failure). The probability function of the binomial distribution is given below:

$$f(x|\theta) = \binom{N}{x} \theta^x (1-\theta)^{N-x}$$

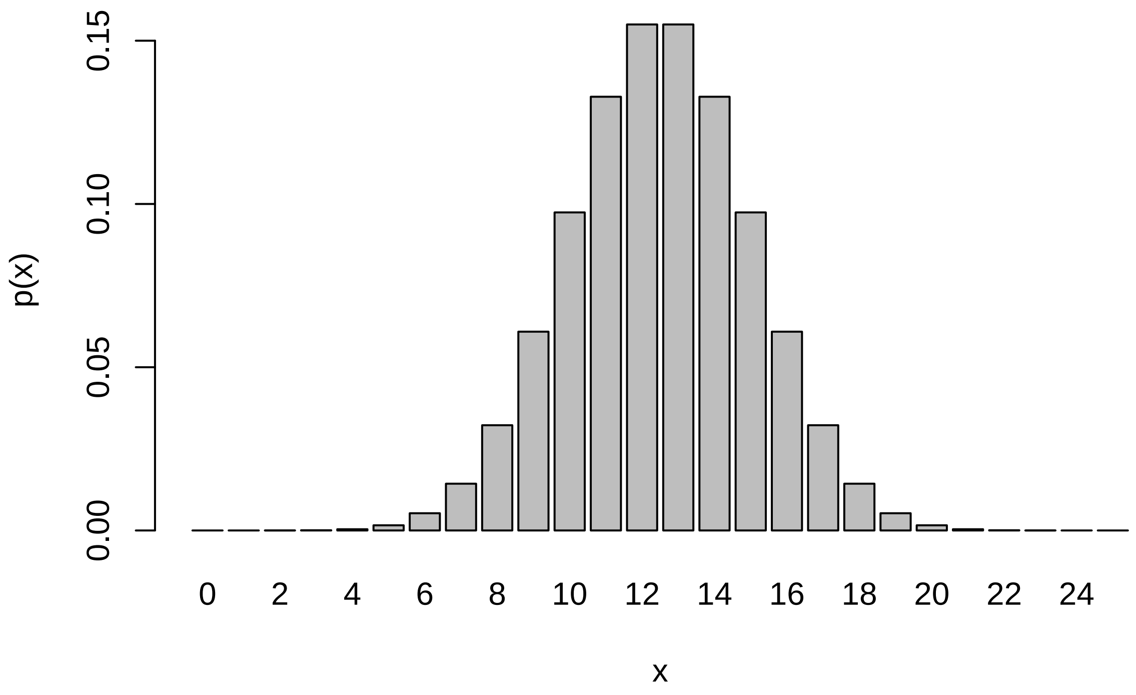
where,

x = Number of successes and θ = probability of success.

This function above gives us the probability of data, given a specific parameter.

Let us see the barplot for probability function where $N = 25$ trials and fixed parameter of $\theta = 0.5$. In each of these 20 trials, the probability of landing heads is 50%.

```
barplot(dbinom(0:25,size=25,prob=0.5),
        names.arg = 0:25,
        ylab = "p(x)",
        xlab = "x"
        )
```



The plot above represents the probability of getting each possible number of heads in 25 outcomes. We can see that most of the occurrences are somewhere between 12 and 14. The most probable outcome is 12 and 14 heads. It tells the probability of each discrete outcome.

Now, let us look at the function which gives the likelihood of a range of parameters, given a specific data point.

$$f(\theta|x) = \binom{N}{x}\theta^x(1-\theta)^{N-x}$$

Now, our input is our parameters instead of data and this is called our likelihood function. Example: Suppose we observe 10 successes in 25 trials. Here, we are not asking about the probability of 10 successes but instead we are asking what is the parameter that would have given us 10 successes.

Let us see the plot of above explained situation i.e. Suppose that in a sequence of 25 coin flips, we observe 10 successes ('heads'). What is the maximum likelihood estimate for θ?. Assumption: Assume we have a binomial model i.e.,

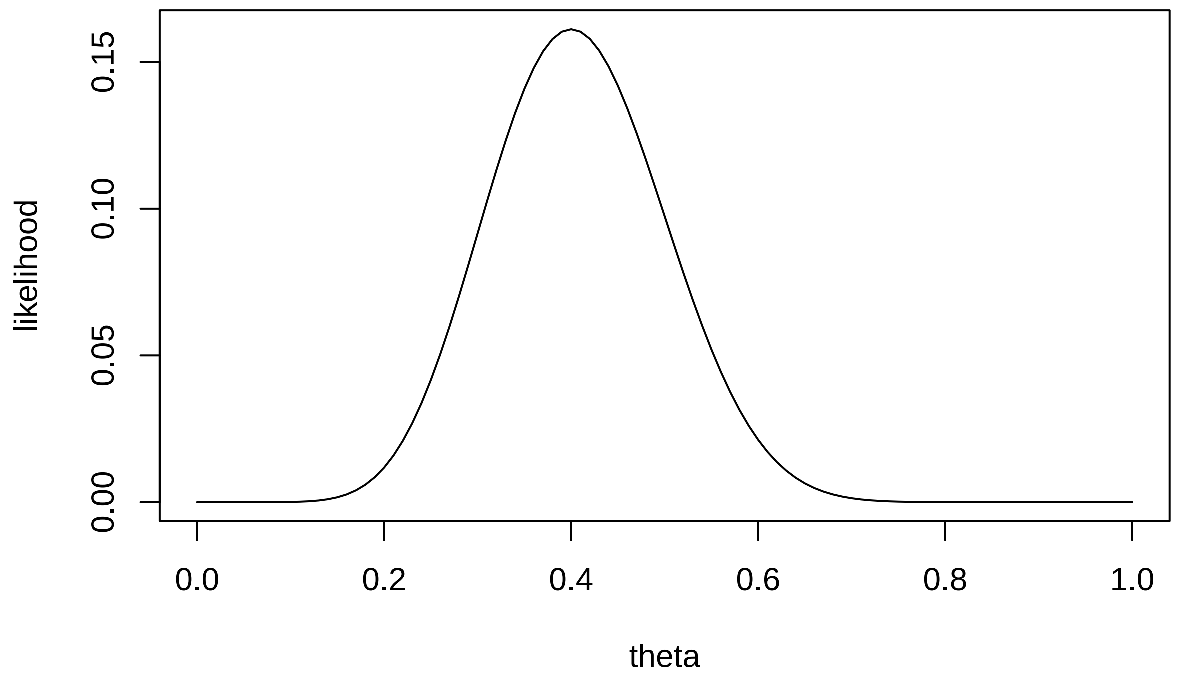
$$x \sim Binomial(\theta, n = 25)$$

```
# Defining likelihood function
negative_log_likelihood_binom = function(data,par){
  return(-log(dbinom(data,size=25,prob=par)))
}

# finding parameters that maximizes the likelihood function
```

```
optim(par = 0.5, fn = negative_log_likelihood_binom, data=10) # par=0.5 is the initial guess of the par
```

```
## Warning in optim(par = 0.5, fn = negative_log_likelihood_binom, data = 10): one-dimensional optimiza
## use "Brent" or optimize() directly

## $par
## [1] 0.4
##
## $value
## [1] 1.82537
##
## $counts
## function gradient
##       28       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
theta = seq(from = 0, to = 1, by = 0.01) # theta ranges from 0 to 1.
plot(theta, dbinom(x=10,size=25,prob=theta),
     type ="l", ylab = "likelihood")
```



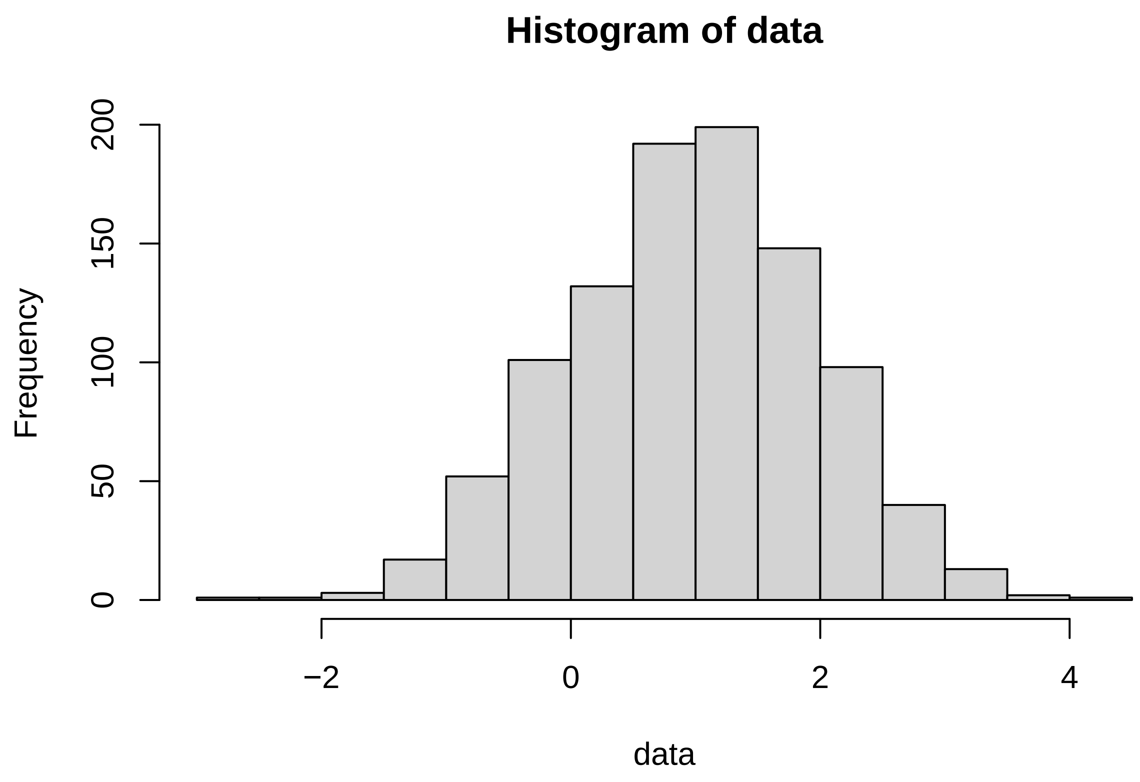
The plot above is telling us the likelihood of observing those 10 successes over the range of possible values of θ. We can see from the plot that as theta approaches 0.4, the likelihood goes up. We can ask the question that what value of θ is most likely, given the data that we observed. We can see that our optimized parameter i.e., MLE = par = 0.4.

Normal Model

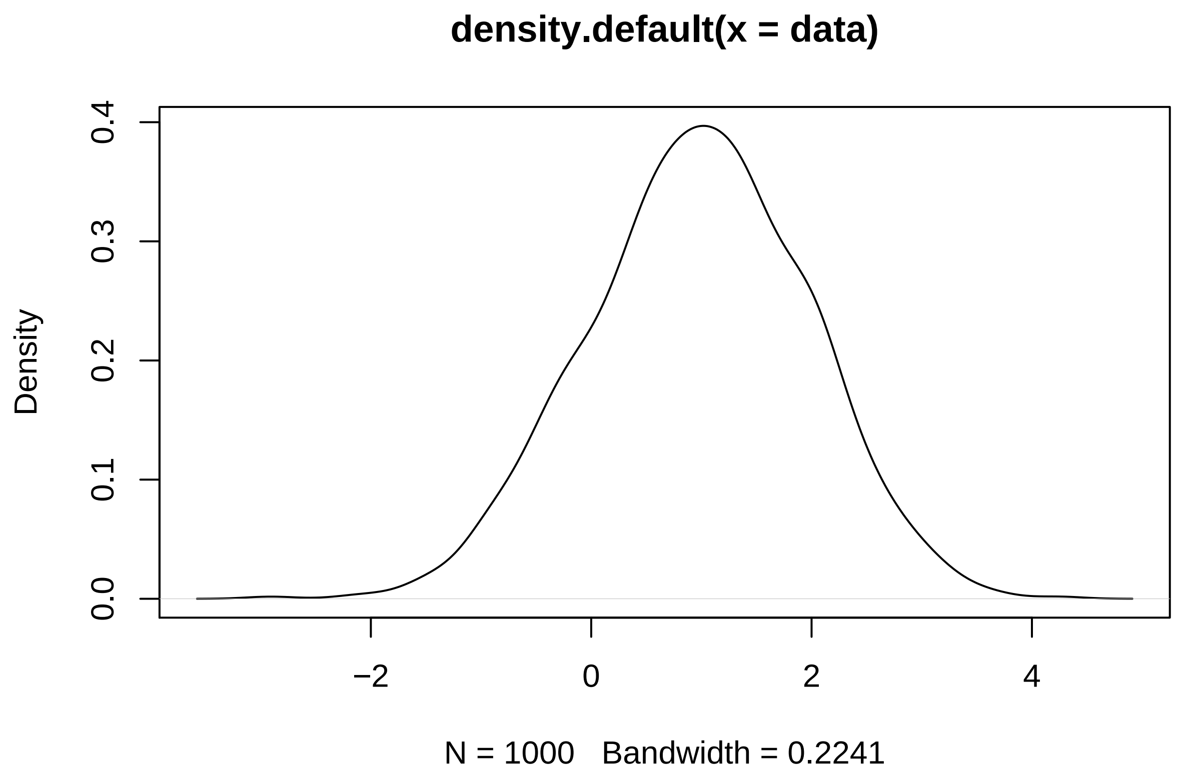
In this example, we are going to fit the data with a normal model and find the Maximum Likelihood Estimation's for mean and standard deviation. Suppose we have 1000 observations. Assumption: We have a normal model with parameters μ, and σ

$$x \sim Normal(\mu, \sigma)$$

```
data = rnorm(1000,1,1) # Generate 1000 random observations with mean one and standard deviation 1.
hist(data)
```



```
plot(density(data))
```



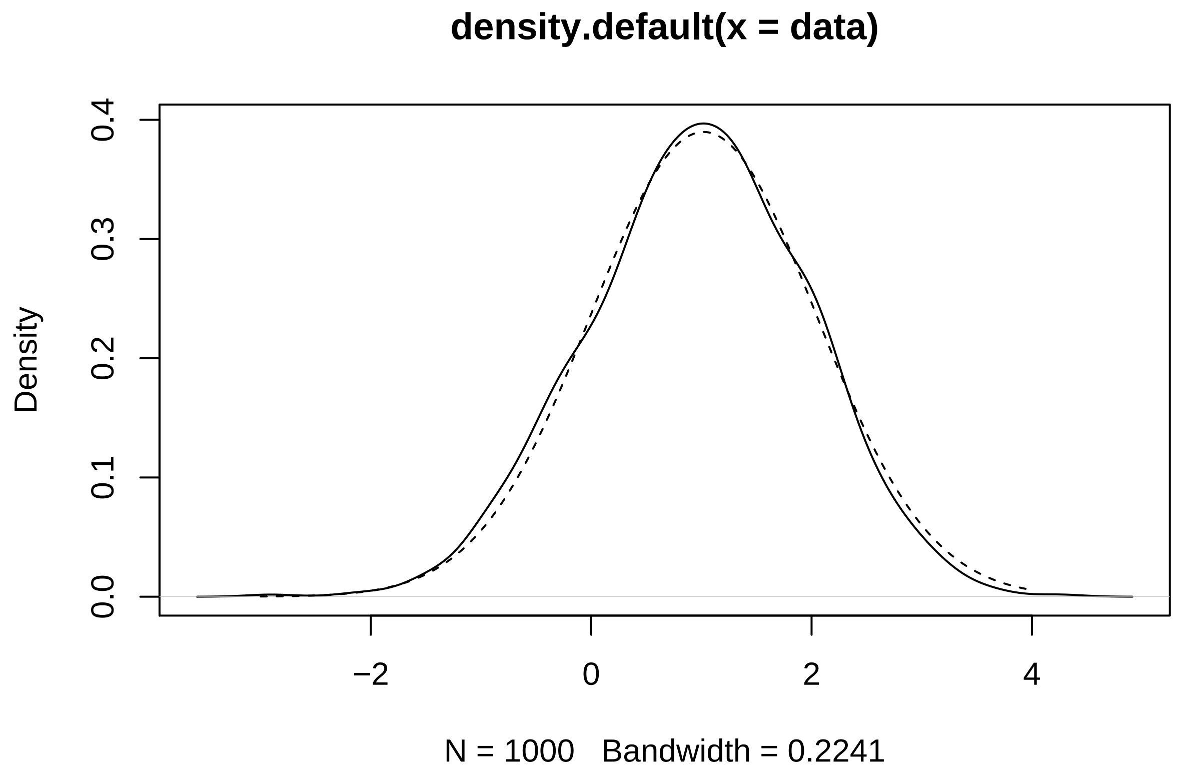
```
# Defining likelihood function
negative_log_likelihood_normal = function(data,par){ # our parameter par is two dimensional vector with
  return(-sum(log(dnorm(data,mean=par[1],sd=par[2]))))
}

# finding parameters that maximizes the likelihood function
optim(par=c(1,1), fn = negative_log_likelihood_normal, data = data)
```

```
## $par
## [1] 0.9703582 0.9905817
##
## $value
## [1] 1409.513
##
## $counts
## function gradient
##       47       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

From the results, we can see that maximum likelihood estimates for parameters mean and sigma is 1.02 and

1.023 which is close to theoretical mean of 1 an theoretical standard deviation of 1. We can see how well our model "fits the data by performing the below steps.

```r
x = seq(-3,4,0.01)
plot(density(data))
lines(x,dnorm(x,mean=1.020090,sd=1.023422),lty=2)
```

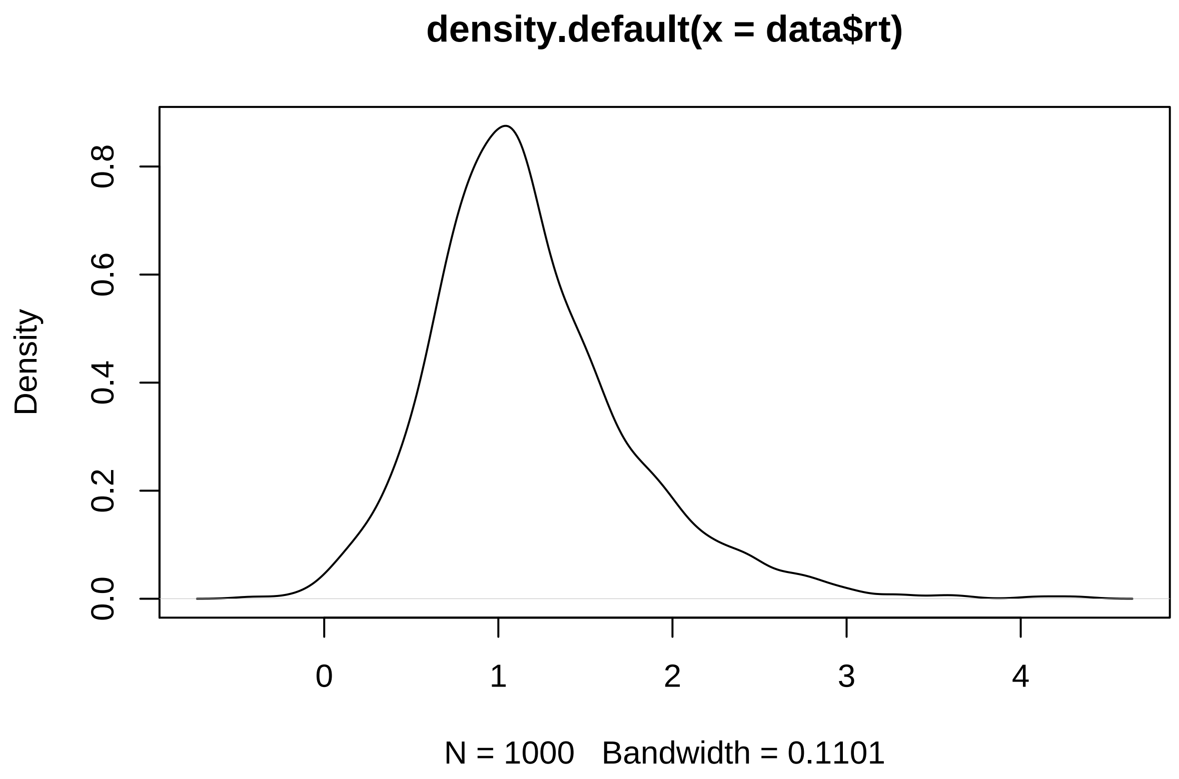


We can see that our estimate fits the data pretty well.

ex-Gaussian Model

Consider the data set of 1000 response times (RTs) in a mental arithmetic task, followed by a plot of the data.

```r
data = read.csv("https://git.io/v58yI")
plot(density(data$rt))
```



We can see that it is no longer symmetric about the peak that is it is no longer normal. It has a positive skewness. We would not be able to fit in the normal model to this data. Distributions with long rightward tails can be modeled by an ex-Gaussian distribution which is the combination of an exponential and normal distribution. We can fit the response time as ex-Gaussian with three parameters,

$$rt \sim exGaussian(\mu, \sigma, \tau)$$

where,

μ = mean of normal component

σ = standard deviation of normal component

τ = rate of exponential component

Ex-Gaussian distribution is not built-in R so we need to create one ourselves.

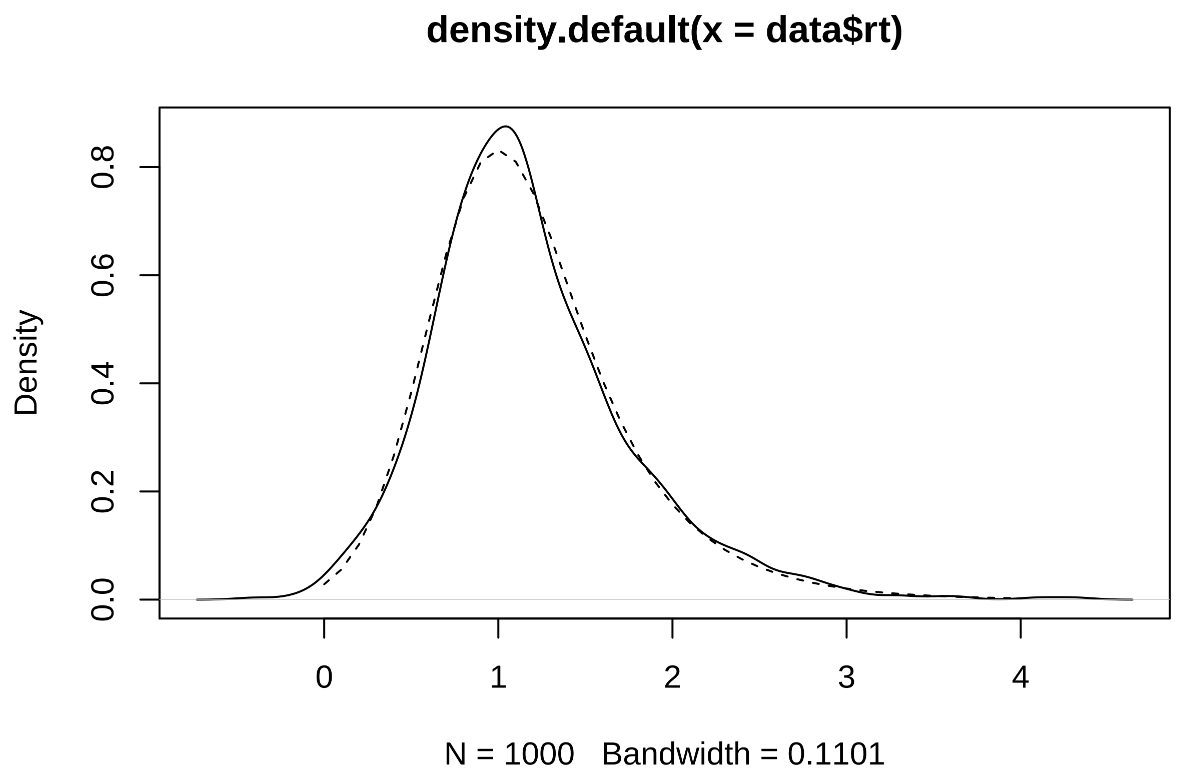
```
dexg = function(x,mu,sigma,tau){
  return((1/tau)*exp((sigma^2/(2*tau^2))-(x-mu)/tau)*pnorm((x-mu)/sigma-(sigma/tau)))
}

# Defining likelihood function
negative_log_likelihood_exg = function(data,par){
  return(-sum(log(dexg(data,
                       mu = par[1],
                       sigma = par[2],
                       tau = par[3]))))
}
```

```r
# finding parameters that maximizes the likelihood function
optim(par=c(0,0.1,0.1), fn=negative_log_likelihood_exg,data=data$rt)
```

```
## $par
## [1] 0.7151246 0.3363003 0.4646291
##
## $value
## [1] 794.2717
##
## $counts
## function gradient
##      150       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Now, let us check the fit of the model.

```r
x = seq(0,4,0.1)
plot(density(data$rt))
lines(x,dexg(x,mu=0.7151246,sigma=0.3363003,tau=0.4646291),lty=2)
```



We can see from the plot that this is the exceptionally good fit. The dashed line is the ex-Gaussian curve and the solid line is the actual data.

ARMA Models

Based on the given observations, we can find the appropriate values of parameters using different methods such as Moment estimates, Least Square estimates, Prediction error estimates (Conditioned, Unconditioned) and Maximum Likelihood estimates(Conditioned, Unconditioned).

Let us look at the time series for AAPL from 2020/01/01 to 2021/01/01. Usong the autocorrelation function, we know

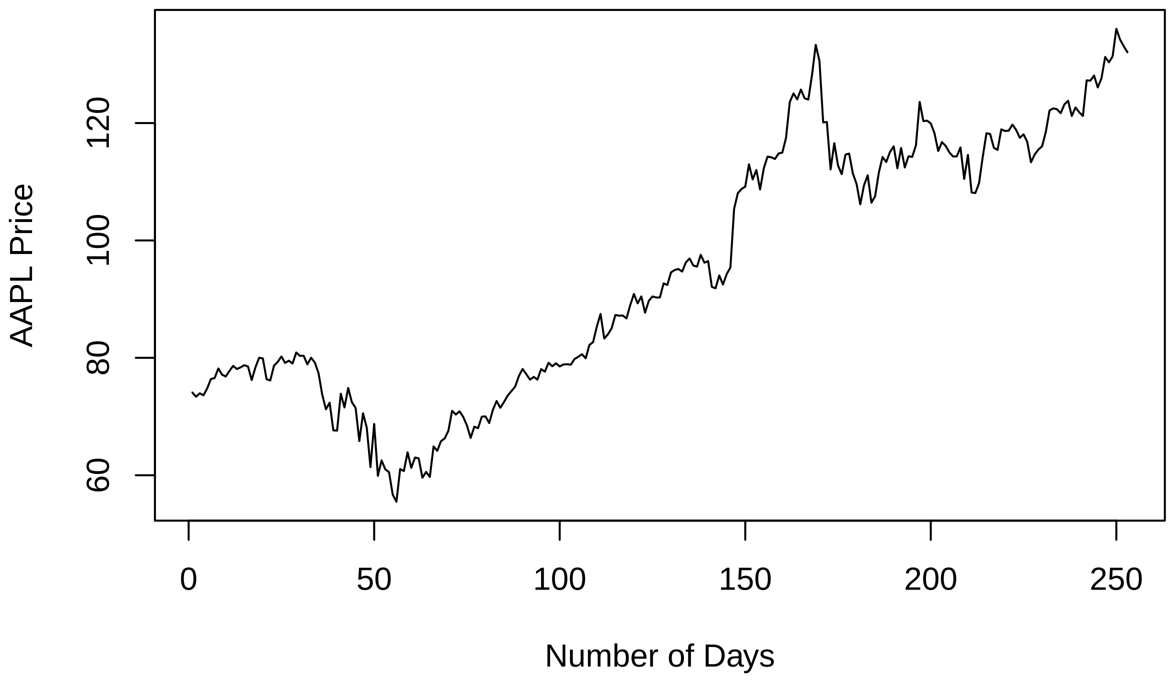
$$\hat{y}_{t+1|t} = a_1 y_t + a_2 y_{t-1}$$

and we should select a_1 and a_2 such that the sum of squared prediction errors is minimized.

```
library(tidyquant)
```

```
## Warning: package 'tidyquant' was built under R version 4.0.5

## Loading required package: lubridate

## Warning: package 'lubridate' was built under R version 4.0.3

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

## Loading required package: PerformanceAnalytics

## Warning: package 'PerformanceAnalytics' was built under R version 4.0.5

## Loading required package: xts

## Warning: package 'xts' was built under R version 4.0.2

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 4.0.2

##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

##
## Attaching package: 'xts'

## The following objects are masked from 'package:data.table':
##
##     first, last

##
## Attaching package: 'PerformanceAnalytics'

## The following object is masked from 'package:graphics':
##
##     legend

## Loading required package: quantmod

## Warning: package 'quantmod' was built under R version 4.0.3

## Loading required package: TTR

## Warning: package 'TTR' was built under R version 4.0.2

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

## == Need to Learn tidyquant? ====================================================
## Business Science offers a 1-hour course - Learning Lab #9: Performance Analysis & Portfolio Optimiza
## </> Learn more at: https://university.business-science.io/p/learning-labs-pro </>
```

```r
av_api_key('PQQC4A86NCLAXUSQ')
aapl = tq_get("AAPL", from = "2020-01-01", to = "2021-01-01")
plot(aapl$adjusted, type = "l", ylab = "AAPL Price", xlab = "Number of Days")
```



In response to the Brockwell and Davis textbook, we will write the one-step forecast as:

$$\hat{y}_{t+1|t} = -\phi_1 y_t - \phi_2 y_{t-1}$$

Suppose, we have the number of observations $y_1, y_2, ..., y_n$ and errors $e_{t+1|t} = y_{t+1} - \hat{y}_{t+1|t} = y_{t+1} - (-\phi_1 y_t - \phi_2 y_{t-1}$

We can do it for all of the time points. Let us start from time point 3.

$$\begin{aligned} e_{3|2} &= y_3 + \phi_1 y_2 + \phi_2 y_1 \\ e_{4|3} &= y_4 + \phi_1 y_3 + \phi_2 y_2 \\ e_{5|4} &= y_5 + \phi_1 y_4 + \phi_2 y_3 \\ &\vdots \\ e_{n|n-1} &= y_n + \phi_1 y_{n-1} + \phi_2 y_{n-2} \end{aligned}$$

We can write above equation in more smart way i.e. in matrix form

$$\begin{bmatrix} y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} -y_2 & -y_1 \\ \vdots & \\ -y_{n-1} & -y_{n-2} \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_2 \end{bmatrix} + \begin{bmatrix} e_{3|2} \\ \vdots \\ e_{n|n-1} \end{bmatrix}$$

Or, we can just write

$$Y = X\theta + \epsilon$$

To minimize the sum of the squared 1-step prediction errors $\epsilon^T\epsilon$, we use the result for the GLM:

$$\hat{\theta} = (X^TX)^{-1}X^TY$$

with,

$$X = \begin{bmatrix} -y_2 & -y_1 \\ \vdots & \\ -y_{n-1} & -y_{n-2} \end{bmatrix}$$

and

$$Y = \begin{bmatrix} y_3 \\ \vdots \\ y_n \end{bmatrix}$$

We have the variance of the estimator asymptotically: $V(\hat{\theta}) = \sigma_\epsilon^2(X^TX)^{-1}$.

This method is called the least square-estimator for dynamical systems. It is also in the class of prediction error methods since it minimizes the sum of squared 1-step prediction errors.

Now the question is, how do we generalize this to AR(p)- models? We have to do is that we need P columns for X matrix, we have y(p) in the first column that means the first observation we can use is $y(p+1)$ for Y matrix. So, we have $n-p$ observations that we can use for AR(p) model.

Let us look at simple example with six observations:

```
observations = c(-3.11,-2.95,-1.80,-2.05,-1.95,-0.80)
N = length(observations)
Y = observations[3:N] # We want to fit the AR(2) model
X = cbind(-observations[2:(N-1)],-observations[1:(N-2)]) # design matrix
solve(t(X) %*% X, t(X) %*% Y) # Estimates
```

```
##             [,1]
## [1,] -0.1471890
## [2,] -0.5241497
```

The above mentioned is least squared method and it works well when we have a pure autoregressive model. But, when we have a moving average part in the model as well then we need to do something different that is where Maximum Likelihood Estimation comes in.

Maximum Likelihood Estimation for ARMA(p,q) process

Consider $ARMA(p,q)$ process:

$$Y_t + \phi_1 Y_{t-1} + ... + \phi_p Y_{t-p} = \epsilon_t + \theta_1\epsilon_{t-1} + ... + \theta_q\epsilon_{t-q}$$

The parameters for AR part are $\phi's$ and the parameters for moving average parts are $\theta's$. Now, I will combine all the parameters into a single set of vectors as,

$$\theta^T = (\phi_1, ..., \phi_p, \theta_1, ..., \theta_q)$$

and Y_t^T as a vector containing all the observations upto time T,

$$Y_t^T = (Y_t, Y_{t-1}, ..., Y_1)$$

As we said before, the likelihood function is the joint probability distribution function for all observations for given values of θ and σ_ϵ^2, i.e.,

$$L(Y_n : \theta, \sigma_\epsilon^2) = f(Y_n|\theta, \sigma_\epsilon^2)$$

given the observations Y_n we estimate θ and σ_ϵ^2 as the values for which the likelihood is maximized.

Some Assumptions:

The random variables $Y_n|Y_{n-1}$ only contains ϵ_n as a random component.ϵ_n is a white noise process at time n and does therefore does not depend on anything.We therefore know that the random variables $Y_n|Y_{n-1}$ are independent,hence:

$$f(Y_n|\theta, \sigma_\epsilon^2) = f(Y_n|Y_{n-1}, \theta, \sigma_\epsilon^2) f(Y_{n-1}|\theta, \sigma_\epsilon^2)$$

We can repeat the above arguments and have the likelihood function defined below:

$$L(Y_n : \theta, \sigma_\epsilon^2) = \Big(\prod_{t=p+1}^{N} f(Y_t|Y_{t-1}, \theta, \sigma_\epsilon^2) \Big) f(Y_p|\theta, \sigma_\epsilon^2)$$

It turns out that the estimates obtained using the conditional likelihood function:

$$L(Y_n : \theta, \sigma_\epsilon^2) = \prod_{t=p+1}^{N} f(Y_t|Y_{t-1}, \theta, \sigma_\epsilon^2)$$

is almost the same estimates as the exact likelihood function when many observations are available. For small samples, there is a slight differences. The function *arima* in R calculates the exact estimates per default.

Evaluating Conditional Likelihood Function

We are interested in finding the conditional densities given specified values of parameters θ and σ_ϵ^2 . The mean of the random variable $Y_t|Y_{t-1}$ is the one step forecast $\hat{Y}_{t|t-1}$. The prediction error $\epsilon_t = Y_t - \hat{Y}_{t|t-1}$ has variance σ_ϵ^2. We assume that the process is Gaussian:

$$f(Y_t|Y_{t-1}, \theta, \sigma_\epsilon^2) = \frac{1}{\sigma_\epsilon \sqrt{2\pi}} \exp\Big(-\frac{1}{2\sigma_\epsilon^2} (Y_t - \hat{Y}_{t|t-1}(\theta))^2 \Big)$$

So, we have the likelihood function as:

$$L(Y_n : \theta, \sigma_\epsilon^2) = (\sigma_\epsilon^2 2\pi)^{-\frac{n-p}{2}} \exp\Big(-\frac{1}{2\sigma_\epsilon^2} \sum_{t=p+1}^{n} \epsilon_t^2(\theta) \Big)$$

The conditional ML- estimate, $\hat{\theta}$, is a prediction error estimate since it is obtained by minimizing,

$$S(\theta) = \sum_{t=p+1}^{N} \epsilon_t^2(\theta)$$

By differentiating with respect to σ_ϵ^2, we can show that ML-estimate of σ_ϵ^2 is:

$$\hat{\sigma}_\epsilon^2 = \frac{S(\hat{\theta})}{n-p}$$

Note: p is the order of the AR part.

The estimate $\hat{\theta}$ is asymptotically unbiased and efficient and the variance-covariance matrix is approximately

$$2\sigma_\epsilon^2 H^{-1}$$

where H contains the second order partial derivatives of $S(\theta)$ around the minimum.

Maximum Likelihood estimates using PE-method

Suppose we have 1-step prediction model defined as follows:

$$\hat{Y}_{t|t-1} = -\phi_1 Y_{t-1} - ... - \phi_p Y_{t-p} + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q}$$

If we use (condition on) $\epsilon_p = \epsilon_{p-1} = ... = \epsilon_{p+1-q} = 0$ then we can find

$$\hat{Y}_{p+1|p} = -\phi_1 Y_p - ... - \phi_p Y_1 + \theta_1 \epsilon_p + ... + \theta_q \epsilon_{p+1-q}$$

which will give us $\epsilon_{p+1} = Y_{p+1} - \hat{Y}_{p+1|p}$ and then we can calculate $\hat{Y}_{p+2|p+1}$ and ϵ_{p+2} and so on until we have all the 1-step prediction error we need. We use the numerical optimization to find the parameters which minimizes the sum of squared of the prediction errors.

#AAPL example from above

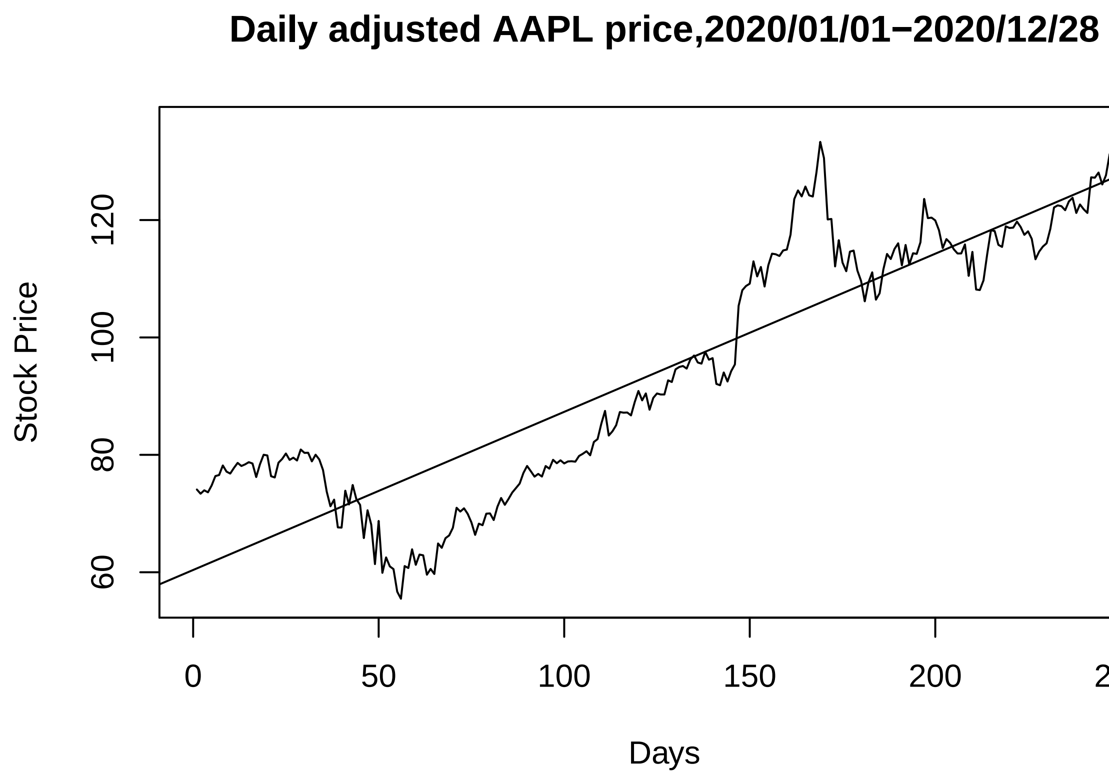
```
aapldata = ts(data=aapl$adjusted)
is.ts(aapldata)
```

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

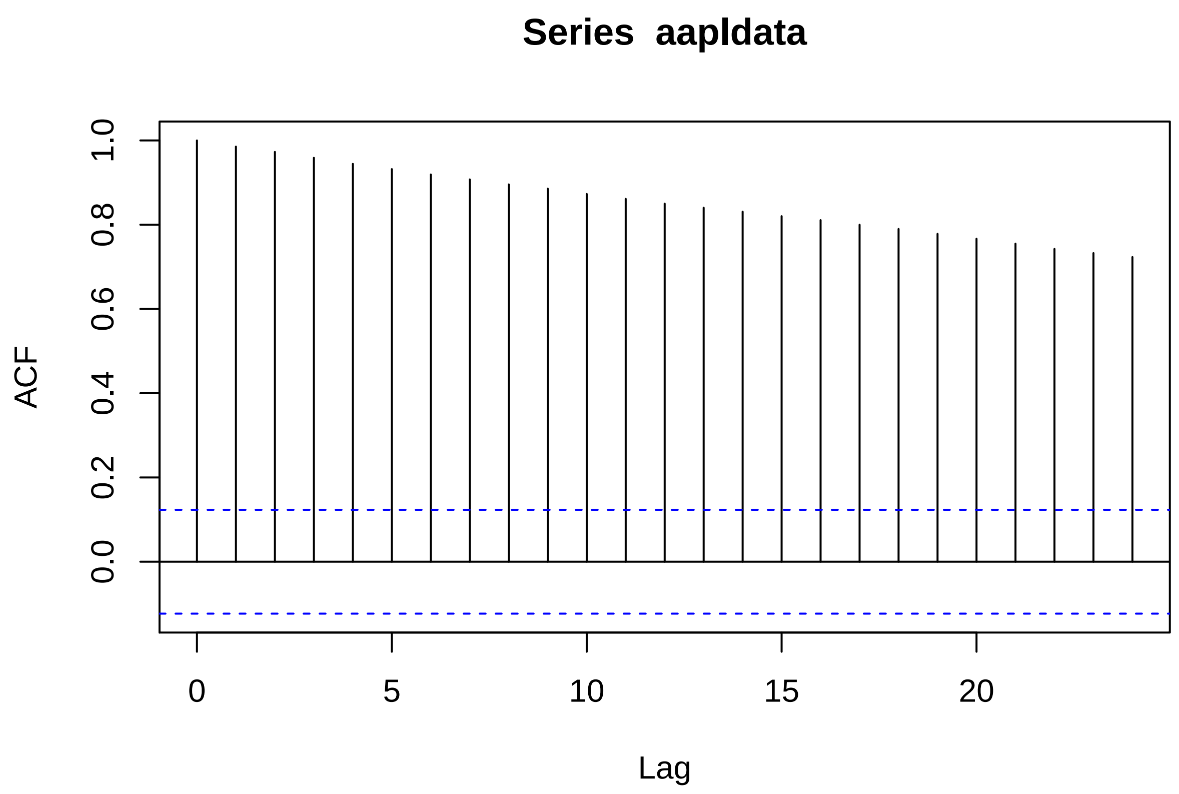
```
summary(aapldata)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   55.48   76.39   90.88   94.60  115.01  136.05
```

```
ts.plot(aapldata,xlab="Days",ylab="Stock Price",main="Daily adjusted AAPL price,2020/01/01-2020/12/28")
abline(reg=lm(aapldata~time(aapldata)))
```



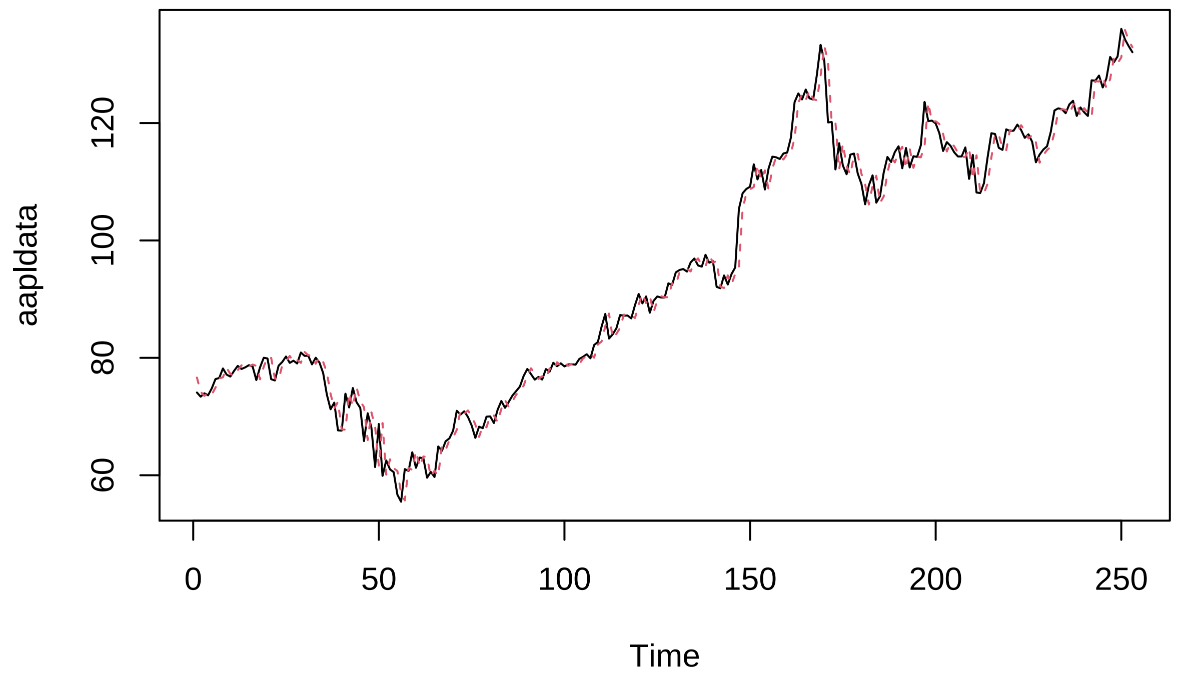
```
acf(aapldata)
```



```
AR = arima(x = aapldata, order = c(1,0,0))
print(AR)
```

```
##
## Call:
## arima(x = aapldata, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.9949    99.2505
## s.e.  0.0052    19.9510
##
## sigma^2 estimated as 6.685:  log likelihood = -601.61,  aic = 1209.21
```

```
ts.plot(aapldata)
AR_fit = aapldata - residuals(AR)
points(AR_fit, type = "l", col=2, lty=2)
```



```r
# Using predict() to make a one-step forecast
predict_AR = predict(AR)

# Obtaining the one tep forecast using $pred[1]
predict_AR$pred[1]
```

```
## [1] 131.901
```

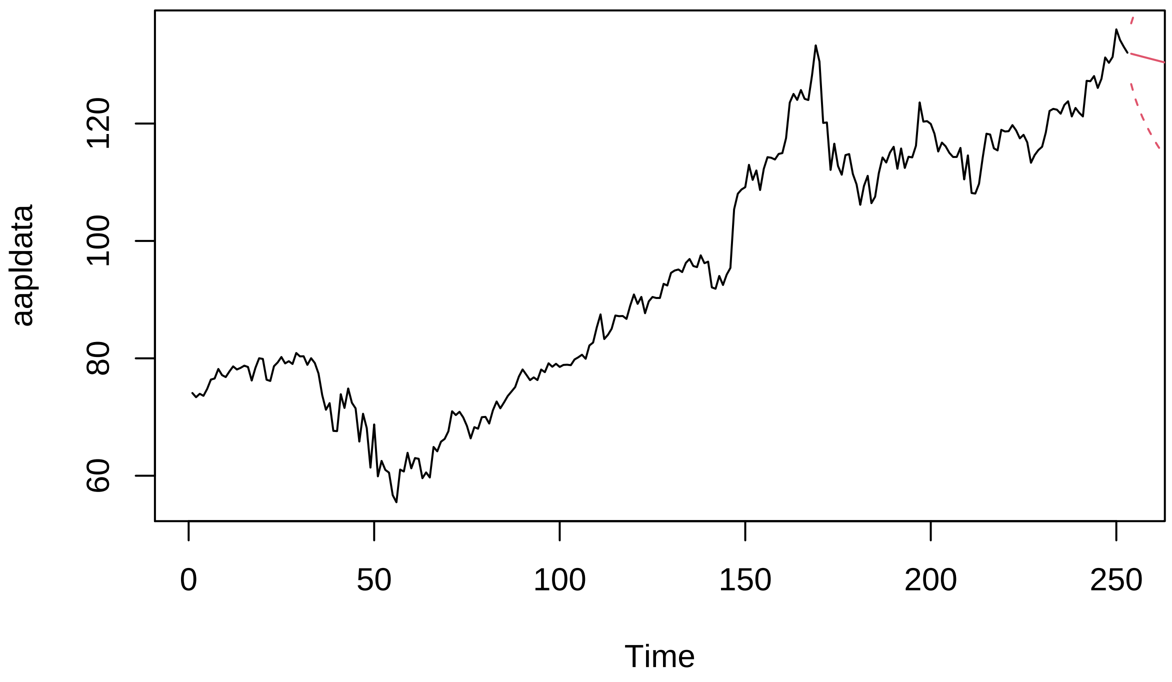
```r
# Alternatively using predict to make 1-step thorugh 10 step forecasts
predict(AR, n.ahead=10)
```

```
## $pred
## Time Series:
## Start = 254
## End = 263
## Frequency = 1
##  [1] 131.9010 131.7335 131.5668 131.4009 131.2359 131.0718 130.9085 130.7460
##  [9] 130.5844 130.4236
##
## $se
## Time Series:
## Start = 254
## End = 263
## Frequency = 1
##  [1] 2.585455 3.647015 4.455236 5.131325 5.722369 6.252587 6.736408 7.183258
##  [9] 7.599705 7.990545
```

```
# Plotting the aapl data series plus the forecasts and 95% prediction intervals
ts.plot(aapldata)
AR_forecast = predict(AR,n.ahead=10)$pred
AR_forecast_se = predict(AR,n.ahead=10)$se
points(AR_forecast, type = "l", col = 2)
points(AR_forecast - 2*AR_forecast_se,type = "le",col=2,lty=2)
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): plot type 'le' will be
## truncated to first character
```

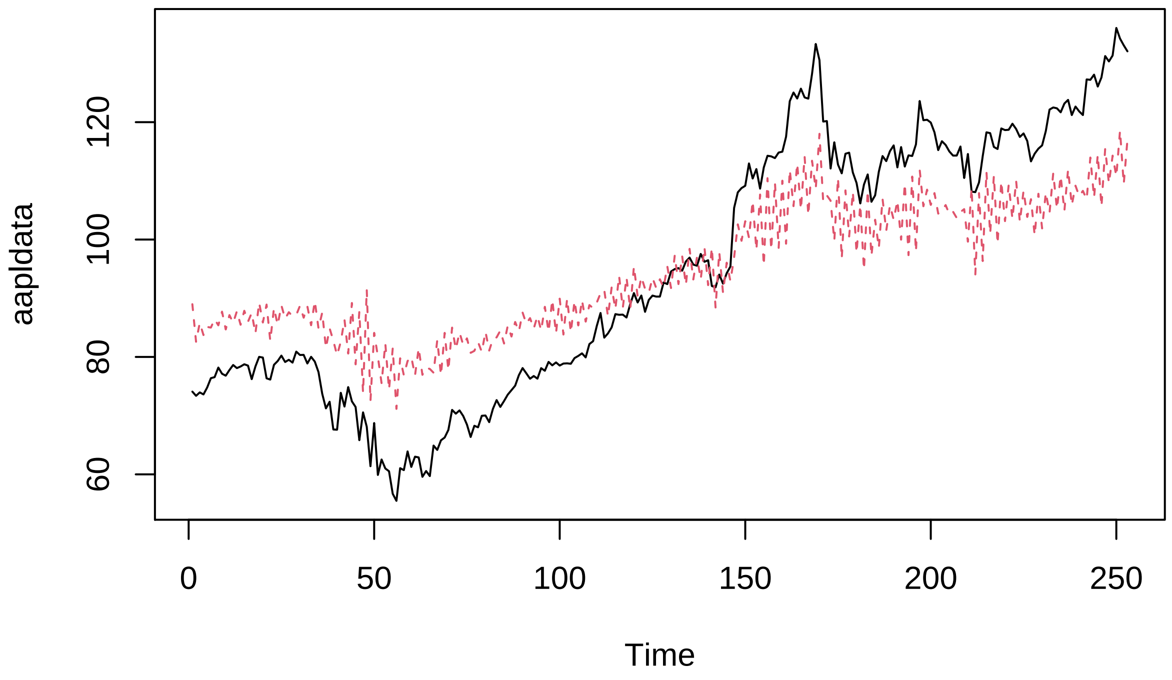
```
points(AR_forecast + 2*AR_forecast_se, type="l",col=2,lty=2)
```



```
# Fitting the MA model to aapldata
MA = arima(aapldata, order = c(0,0,1))
print(MA)
```

```
##
## Call:
## arima(x = aapldata, order = c(0, 0, 1))
##
## Coefficients:
##          ma1  intercept
##       0.9475    94.6162
## s.e.  0.0168     1.4350
##
## sigma^2 estimated as 137.9:  log likelihood = -983.32,  aic = 1972.65
```

```
#plotting the series along with the MA fitted values
ts.plot(aapldata)
MA_fit <- aapldata - resid(MA)
points(MA_fit, type = "l", col = 2, lty = 2)
```



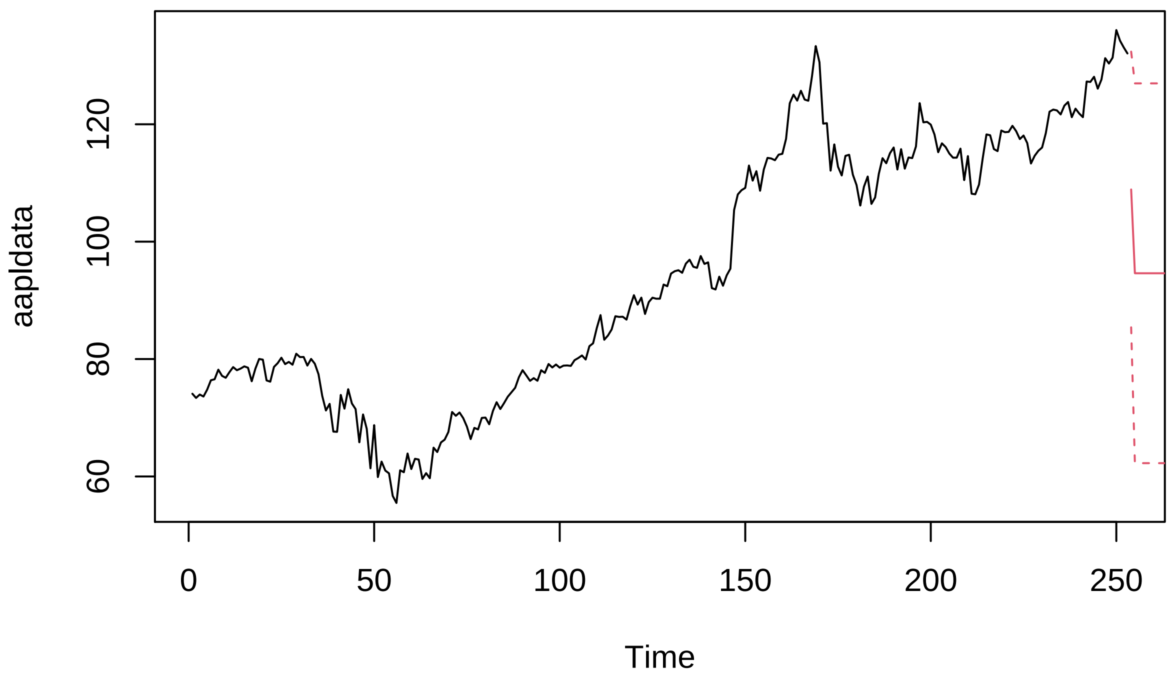
```
#Making a 1-step forecast based on MA
predict_MA <- predict(MA)

#Obtaining the 1-step forecast using $pred[1]
predict_MA$pred[1]
```

```
## [1] 108.8919
```

```
#Alternately Making a 1-step through 10-step forecast based on MA
predict(MA,n.ahead=10)
```

```
## $pred
## Time Series:
## Start = 254
## End = 263
## Frequency = 1
##  [1] 108.89192  94.61625  94.61625  94.61625  94.61625  94.61625  94.61625
##  [8]  94.61625  94.61625  94.61625
##
## $se
## Time Series:
## Start = 254
## End = 263
## Frequency = 1
##  [1] 11.74246 16.17661 16.17661 16.17661 16.17661 16.17661 16.17661 16.17661
##  [9] 16.17661 16.17661
```

```
#Plotting the aapldata series plus the forecast and 95% prediction intervals
ts.plot(aapldata)
MA_forecasts <- predict(MA, n.ahead = 10)$pred
MA_forecast_se <- predict(MA, n.ahead = 10)$se
points(MA_forecasts, type = "l", col = 2)
points(MA_forecasts - 2*MA_forecast_se, type = "l", col = 2, lty = 2)
points(MA_forecasts + 2*MA_forecast_se, type = "l", col = 2, lty = 2)
```



```
# Find correlation between AR_fit and MA_fit
cor(AR_fit, MA_fit)
```

```
## [1] 0.9580514
```

```
# Find AIC of AR
AIC(AR)
```

```
## [1] 1209.215
```

```
# Find AIC of MA
AIC(MA)
```

```
## [1] 1972.649
```

```
# Find BIC of AR
BIC(AR)
```

```
## [1] 1219.815
```

```
# Find BIC of MA
BIC(MA)
```

```
## [1] 1983.249
```

```
# Given lower values of AIC and BIC in AR model, we should choose AR model for time series analysis of
```